

Magento® U



Magento® U

Contents – Unit Six

About This Guide.....	viii
1. Unit Six Home Page	1
1.1 Fundamentals of Magento 2 Development - Unit Six	1
1.2 Unit Six Home Page	2
2. Adminhtml Intro.....	3
2.1 Adminhtml Introduction.....	3
2.2 Module Topics Adminhtml	4
2.3 Introduction Definition	5
2.4 Introduction Admin Interface	6
2.5 Introduction Admin Controller	7
2.6 Introduction Grids.....	8
2.7 Introduction Forms.....	9
2.8 Introduction System Configuration	10
3. Grids Part 1.....	11
3.1 Grids: Part 1.....	11
3.2 Module Topics Grids	12
3.3 Grids Overview	13
3.4 Grids Adminhtml Grids	14
3.5 Grids Area Locations Diagram	15
3.6 Grids Filters.....	16
3.7 Grids layout.xml.....	17
3.8 Grids Listing UiComponents	18
3.9 Grids Listing UiComponent	19
3.10 ListingUiComponent Overview	20
3.11 Listing Configuration (definitions.xml)	21
3.12 Listing UiComponent Configuration (cms)	22
3.13 Grids Filters.....	23
3.14 Filters Overview	24

3.15 Grids Filters Overview.....	25
3.16 Filters UiComponent.....	26
3.17 Filters Product Listing Example.....	27
3.18 Filters Product Listing Example.....	28
3.19 Filters Filter Pool.....	29
3.20 Reinforcement Exercise (6.3.1)	30
3.21 Grids DataSource	31
3.22 DataSource Overview	32
3.23 DataSource Default DataSource	33
3.24 DataSource Default DataSource Configuration	34
3.25 DataSource Default DataSource	35
3.26 DataSource Catalog Configuration	36
3.27 DataSource Catalog DataProvider	37
3.28 Reinforcement Exercise (6.3.2)	38
4. Grids Part 2.....	39
4.1 Grids: Part 2.....	39
4.2 Module Topics Grids	40
4.3 Grids Grid Indexer.....	41
4.4 Grid Indexer Overview.....	42
4.5 Grid Indexers Schema	43
4.6 Grid Indexer Orders Example 1.....	44
4.7 Grid Indexer Orders Example 2.....	45
4.8 Grid Indexer Orders Example 3.....	46
4.9 Reinforcement Exercise (6.4.1)	47
4.10 Grids Columns.....	48
4.11 Columns Overview.....	49
4.12 Columns definitions.xml	50
4.13 Columns Column JavaScript.....	51
4.14 Columns column.js.....	52
4.15 Columns definitions.xml	54

4.16 Columns Invisible Columns	55
4.17 Columns Invisible Column Configuration	56
4.18 Reinforcement Exercise (6.4.2)	57
4.19 Grids Mass Actions	58
4.20 Mass Actions Overview	59
4.21 Mass Actions Area Location Diagram	60
4.22 Mass Actions definitions.xml	61
4.23 Mass Actions Actions Config Example	62
4.24 Mass Actions JavaScript Module	63
4.25 Reinforcement Exercise (6.4.3)	64
4.26 Grids Paging	65
4.27 Paging Overview	66
4.28 Paging Overview Diagram	67
4.29 Paging default.xml	68
4.30 Paging Example	69
4.31 Paging Default Options	70
4.32 Reinforcement Exercise (6.4.4)	71
5. Forms	72
5.1 Adminhtml Forms	72
5.2 Module Topics Adminhtml Forms	73
5.3 Adminhtml Forms Definition	74
5.4 Adminhtml Forms Format & Fields	75
5.5 Adminhtml Forms Form Container	76
5.6 Adminhtml Forms Magento 2 vs. Magento 1 Approaches	77
5.7 Adminhtml Forms Form UiComponent Structure	78
5.8 Adminhtml Forms Form UiComponent	80
5.9 Form UiComponent Code Definition	81
5.10 Form UiComponent DataProvider Customer Example	82
5.11 Form UiComponent Usage Example	84
5.12 Form UiComponent Magento Block - Buttons	85

5.13 Form UiComponent Magento Block - Buttons	87
5.14 Form UiComponent DataProvider	88
5.15 Form UiComponent DataProvider Customer Example.....	89
5.16 Adminhtml Forms Fieldsets	90
5.17 Fieldsets Overview.....	91
5.18 Fieldsets Code Definition.....	92
5.19 Fieldsets UiComponent Class.....	93
5.20 Fieldsets Customer Form Configuration Example	94
5.21 Fieldsets Containers.....	95
5.22 Fieldsets Interface	96
5.23 Fieldsets Container Definition	97
5.24 Fieldsets Container Configuration Example	98
5.25 Fieldsets Container Class	99
5.26 Adminhtml Forms Form Elements.....	100
5.27 Form Elements Overview	101
5.28 Form Elements Data Types.....	102
5.29 Form Elements Definition Example	103
5.30 Form Elements Generic Field Template	104
5.31 Form Elements Element Template	106
5.32 Form Elements Configuration Example	107
5.33 Form Elements Select Example	108
5.34 Reinforcement Exercise (6.5.1)	109
6. Configuration	110
6.1 Adminhtml: System Configuration - Menu - ACL	110
6.2 Module Topics System Configuration - Menu - ACL.....	111
6.3 System Configuration Definition.....	112
6.4 Sys Config - Menu - ACL System Configuration	113
6.5 System Configuration Overview	114
6.6 System Configuration system.xml	115
6.7 System Configuration Field Attributes.....	116

6.8 System Configuration Field Types	117
6.9 System Configuration Field Children	118
6.10 System Configuration Source Model Configuration	120
6.11 System Configuration Source Model Example	121
6.12 System Configuration Frontend Model Configuration	122
6.13 System Configuration Frontend Model Implementation	123
6.14 Reinforcement Exercise (6.6.1)	126
6.15 Sys Config - Menu - ACL Menu.....	127
6.16 Menu Overview	128
6.17 Menu menu.xml	129
6.18 Reinforcement Exercise (6.6.2)	130
6.19 Sys Config - Menu - ACL ACL.....	131
6.20 ACL Definition	132
6.21 ACL _isAllowed Method.....	133
6.22 ACL _isAllowed Method Implementation Example	134
6.23 ACL Users	135
6.24 ACL Roles	136
6.25 ACL Configuration (Magento/Catalog/etc/acl.xml)	137
6.26 Reinforcement Exercise (6.6.3)	138
6.27 End of Unit Six	139

About This Guide

This guide uses the following symbols in the notes that follow the slides.

Symbol	Indicates...
--------	--------------

	A note, tip, or other information brought to your attention.
--	--

	Important information that you need to know.
--	--

	A cross-reference to another document or website.
--	---

	Best practice recommended by Magento
--	--------------------------------------

1. Unit Six Home Page

1.1 Fundamentals of Magento 2 Development - Unit Six



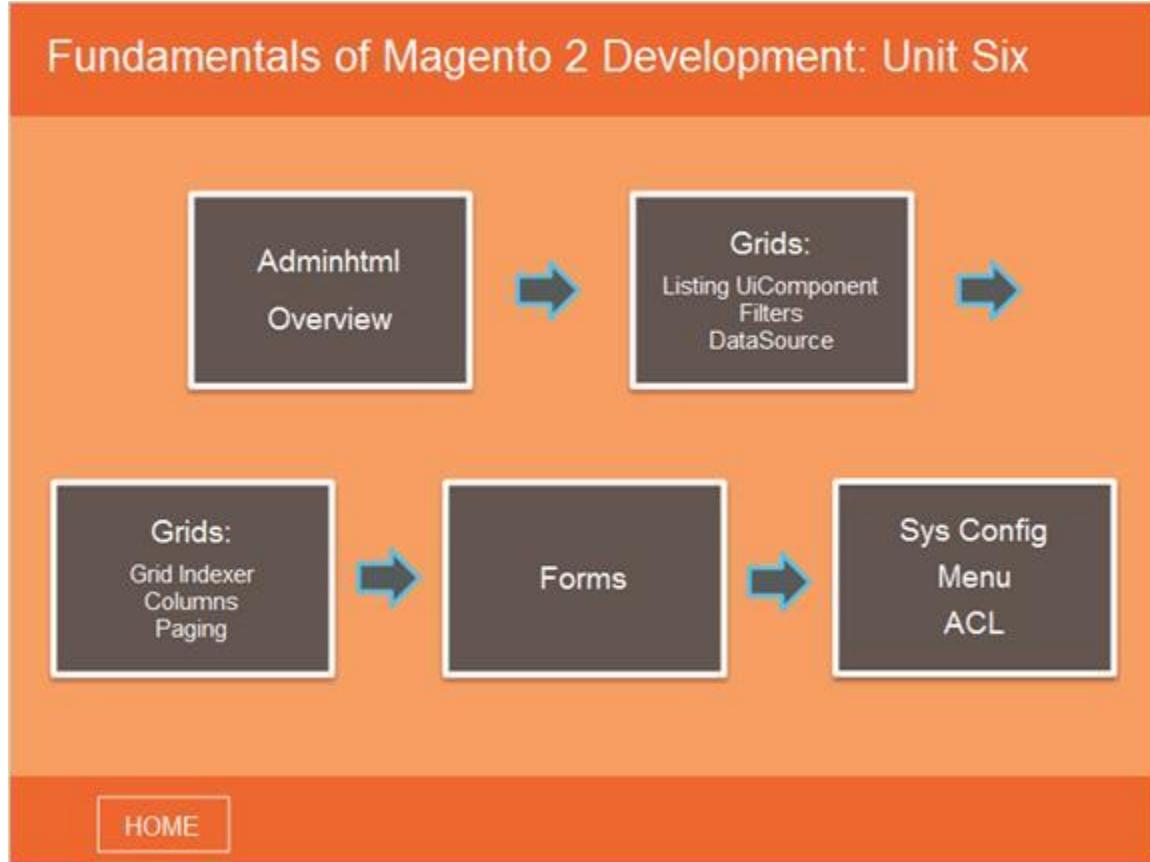
Notes:

Copyright © 2016 Magento, Inc. All rights reserved. 12-01-16

Fundamentals of Magento 2 Development v. 2.1

Software version: Magento 2 v.2.1.0

1.2 Unit Six Home Page



Notes:

Unit Six of the Magento 2 Fundamentals course contains five modules.

The suggested flow of the course is indicated by the arrows. However, you are free to access any of the modules, at any time, by simply clicking the Home button on the bottom of each slide.

2. Adminhtml Intro

2.1 Adminhtml Introduction

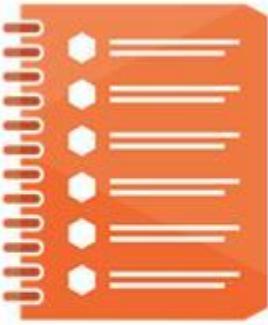
The screenshot shows a slide titled "Adminhtml Introduction". The main content features a blue-toned illustration of a hand holding a magnifying glass over a computer monitor. The monitor displays a gear assembly with arrows pointing left and right, symbolizing configuration or management. Below the illustration is a navigation bar with a "HOME" button and the "Magento U" logo.

Notes:

This course unit provides an overview of the Adminhtml function in Magento 2.

2.2 Module Topics | Adminhtml

Module Topics | Adminhtml



In this module, we will introduce...

- Admin & Admin controller
- Grids
- Forms
- System configuration

[HOME](#) **Magento U**

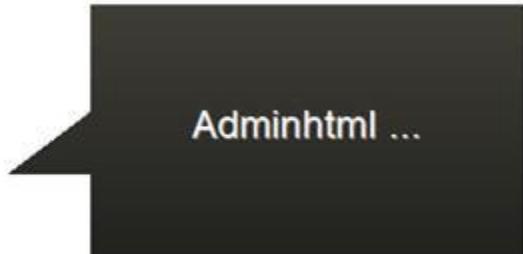
Notes:

In this module, we will introduce you to Adminhtml in Magento 2.0, focusing on:

- Admin and Admin controller
- Grids
- Forms
- System configuration

2.3 Introduction | Definition

Introduction | Definition



- Allows merchants to manage all Magento entities
- Provides a rich interface for configuring the installation

[HOME](#) **Magento U**

Notes:

The Adminhtml function of Magento 2 allows merchants and their staff to manage all Magento entities, such as products, categories, price rules, orders, and so on.

It also provides a rich interface for configuring your Magento installation.

2.4 Introduction | Admin Interface

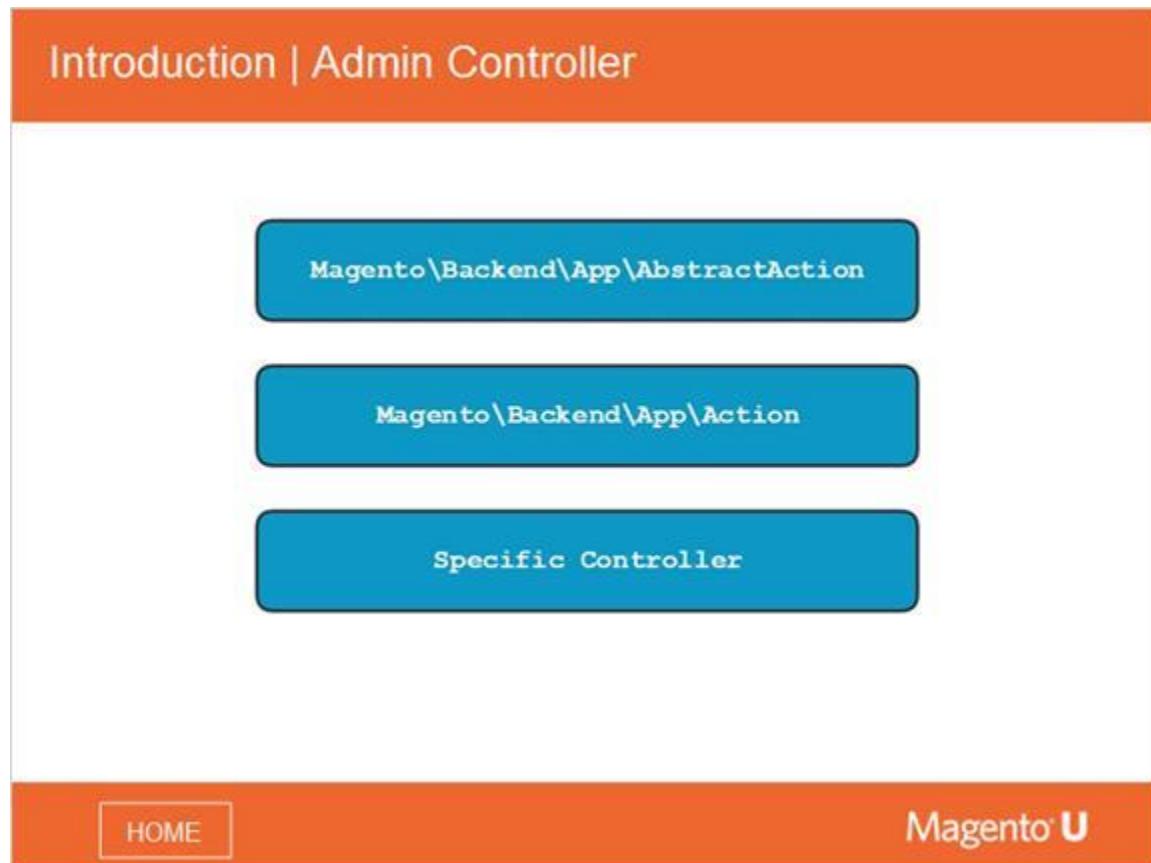
The screenshot shows the Magento 2 Admin Dashboard. At the top, there's an orange header bar with the text "Introduction | Admin Interface". Below this is a white dashboard area with a sidebar on the left containing various icons. The main dashboard section has a grey header "Dashboard" and a "Lifetime Sales" summary: \$29.00. It also displays "Average Order" (\$14.50), "Revenue" (\$0.00), "Tax" (\$0.00), "Shipping" (\$0.00), and "Quantity" (0). A note says "Chart is disabled. To enable the chart, click here." Below this is a "Recent Orders" grid with columns for Customer, Items, and Total. The grid shows five entries for "john Doe" with a total of \$0.00. There are tabs for "Recently" (selected), "Most Viewed Products", "New Customers", and "Customers". At the bottom of the dashboard, there's a message "We couldn't find any records." In the bottom right corner of the dashboard area, there's a "Magento U" logo.

Notes:

Here is the Admin function in Magento 2.

In this unit we will discuss the Admin architecture, specifically grids, forms, system configuration, and ACL.

2.5 Introduction | Admin Controller



Notes:

All admin controllers extend the `Magento\Backend\App\Action` class, which in turn extends `Magento\Backend\App\AbstractAction` (and the latter extends the standard `Magento\Framework\App\Action\Action`).

Each admin controller has to extend this class -- `Magento\Backend\App\Action` -- in order to provide backend-specific functionality, such as secret key processing and ACL mechanism implementation.

2.6 Introduction | Grids

The screenshot shows the 'Catalog' grid in the Magento 2 Admin panel. The grid lists 2048 records found, with sorting and filtering options at the top. The columns include ID, Product ID, Name, Type, Status, Price, Quantity, and Visibility. A 'Mass Actions' dropdown is visible on the left. The data area contains nine product entries, each with a thumbnail, name, type, status, price, quantity, and visibility settings. Annotations with arrows point to the 'button area' (top center), 'sorting' (grid header), 'paging' (bottom right), and 'data area' (grid body). The bottom navigation bar includes 'HOME' and the 'Magento U' logo.

ID	Product ID	Name	Type	Status	Price	Quantity	Visibility	Source	Website	Visible
1	2448001	jean Duffle Bag	Simple Product	Enabled	\$64.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
2	2448002	Shine Shoulder Pack	Simple Product	Enabled	\$12.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
3	2448003	Green Summit Backpack	Simple Product	Enabled	\$36.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
4	2448004	DayFever Messenger Bag	Simple Product	Enabled	\$45.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
5	2448005	Real Puffy Messenger	Simple Product	Enabled	\$45.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
6	2448006	Fusion Backpack	Simple Product	Enabled	\$99.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
7	2448007	Impulse Duffle	Simple Product	Enabled	\$74.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
8	2448008	Vintage Hugo-Bag	Simple Product	Enabled	\$42.00	100.0000	Catalog, Search	Enabled	Main Website	0.00
9	2448009	Compute Track Rose	Simple Product	Enabled	\$21.00	100.0000	Catalog, Search	Enabled	Main Website	0.00

Notes:

Grid is a UIComponent that represents a list of records for a specific entity.

Grids in Magento 2 have significantly changed when compared to Magento 1. They have many new functions, which will be covered in more detail in the Grids module that follows.

2.7 Introduction | Forms

The screenshot shows a customer edit form in the Magento 2 Admin Panel. The form fields are:

- Associate to Website: Main Website
- Group: General
- Disable Automatic Group Change Based on VAT ID: Unchecked
- Prefix: (empty)
- First Name: Veronica
- Middle Name/Initial: (empty)
- Last Name: Costello
- Suffix: (empty)
- Email: @email: roni_cost@dmail.com
- Date Of Birth: (date input field)
- Tax/VAT Number: (empty)

At the bottom left is a "HOME" button, and at the bottom right is the "Magento U" logo.

Notes:

Forms are used to allow editing of a certain entity. They are represented by UiComponents as well as grids.

2.8 Introduction | System Configuration

The screenshot shows the 'System Configuration' page under the 'GENERAL' tab. On the left, there's a sidebar with categories like GENERAL, CATALOG, CUSTOMERS, SALES, and SERVICES. The 'GENERAL' section is expanded, showing sub-options: General, Web, Design, Currency Setup, Store Email Addresses, Contacts, Reports, and Content Management. The main content area is titled 'General' and contains several configuration fields:

- State Options**: A dropdown menu showing 'Timezone' set to 'Eastern European Standard Time (Europe/Kiev)'.
- Locale Options**: A dropdown menu showing 'Locale' set to 'English (United States)'.
- First Day of Week**: A dropdown menu showing 'Sunday'.
- Weekend Days**: A dropdown menu listing days of the week: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. 'Saturday' is highlighted.

At the bottom right of the configuration area is a red 'Save Config' button. Below the configuration area is a navigation bar with 'HOME' and the 'Magento U' logo.

Notes:

As in Magento 1, there is a highly customizable page in Magento 2 where a merchant can add any configuration options required for the installation.

All related values will be stored in a database, and can have different scopes: global, website, store. As this page is used by the merchant, not a developer, all the available options should be clearly presented.

Options that a developer would modify should only be in the XML configuration.

3. Grids Part 1

3.1 Grids: Part 1

Grids: Part 1



The illustration shows a computer monitor with an orange frame. On the screen, there is a user interface element consisting of several orange squares of different sizes arranged in a grid-like pattern. A red cursor arrow points towards one of the squares. The monitor sits on a dark orange base.

HOME

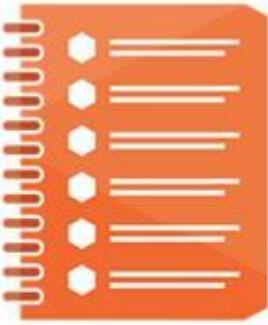
Magento U

Notes:

This course unit discusses the Adminhtml function in Magento 2, starting with Grids.

3.2 Module Topics | Grids

Module Topics | Grids



In this module, we will discuss...

- Listing UiComponent
- Filters
- DataSource

[HOME](#) **Magento U**

Notes:

In this module, we will provide an overview of Adminhtml grids, focusing on the topics of listing UiComponents, filters, DataSource, the grid indexer, columns, and paging.

3.3 Grids | Overview



Notes:

We'll begin with a short review of grids in Magento 2.

3.4 Grids | Adminhtml Grids

Grids | Adminhtml Grids

Adminhtml grids are used to ...

- Render a list of records for a particular entity in the Admin
- Provide an ability to sort, filter, and search in an efficient way through a list of records
- Allow execution of certain events on a group of records (for example: activate, deactivate, delete)

HOME Magento U

Notes:

Adminhtml grids fulfill a number of important functions, such as:

- Render a list of records for a particular entity in admin.
- Provide an ability to sort, filter, and search in an efficient way through a list of records.
- Allow execution of certain events on a group of records (for example: activate, deactivate, delete).

We will look at each of these topics within this section.

3.5 Grids Area Locations Diagram

The screenshot shows the Catalog grid in the Magento 2 admin panel. Several UI areas are highlighted with yellow stars and arrows:

- button area**: Points to the top right corner where there are buttons for "Add Product", "Search", and user profile.
- mass actions**: Points to the "Actions" dropdown menu at the top left.
- filtering**: Points to the filtering section at the top right, which includes dropdowns for "Items", "Default View", and "Columns".
- paging**: Points to the pagination controls at the top right, showing "200 per page" and "1 of 11".
- sorting**: Points to the sorting section at the top center, which includes dropdowns for "Price", "Quantity", "Visibility", "Status", "Attributes", and "Actions".
- data area**: Points to the main grid table containing product data.

The grid table displays the following data:

Thumbnail	Name	Type	Reg.	Price	Quantity	Visibility	Status	Attributes	Actions
	Black Duffle Bag	Simple Product	Bag	24.99\$	100.000	Catalog, Search	Enabled	Main Website	
	Black Shoulder Pack	Simple Product	Bag	24.99\$	100.000	Catalog, Search	Enabled	Main Website	
	Crown Summer Backpack	Simple Product	Bag	24.99\$	100.000	Catalog, Search	Enabled	Main Website	
	Keychain Messenger Bag	Simple Product	Bag	24.99\$	84.00	Catalog, Search	Enabled	Main Website	
	Real Field Messenger	Simple Product	Bag	24.99\$	84.00	Catalog, Search	Enabled	Main Website	
	Fusion Backpack	Simple Product	Bag	24.99\$	84.00	Catalog, Search	Enabled	Main Website	
	Impulse Duffle	Simple Product	Bag	24.99\$	84.00	Catalog, Search	Enabled	Main Website	
	Orange Gym Bag	Simple Product	Bag	24.99\$	100.000	Catalog, Search	Enabled	Main Website	
	Computer Track Bag	Simple Product	Bag	24.99\$	84.00	Catalog, Search	Enabled	Main Website	

At the bottom left is a "HOME" button, and at the bottom right is the "Magento U" logo.

Notes:

On this slide, you can see what a grid looks like in Magento 2 (which is quite different from Magento 1).

The first thing to notice is the filtering function. In Magento 1, there are filters at the top of each column; here in Magento 2, filters are now separated from the data.

You can also see other element areas: buttons, mass actions, paging, sorting, and data.

In this section we will review all these grid components.

In general, grid implementation in Magento 2 has been significantly changed from Magento 1. Now grid is a `UiComponent`, rendered by JavaScript. A developer only has to configure it properly.

3.6 Grids | Filters

The screenshot displays the 'Grids | Filters' interface from the Magento 2 admin panel. It includes the following filter fields:

- Name: Input field
- SKU: Input field
- Type: Select dropdown
- Attribute Set: Select dropdown
- Validity: Select dropdown
- Status: Select dropdown
- Store View: Select dropdown
- Price: Input fields for 'from' and 'to'
- Quantity: Input fields for 'from' and 'to'

At the top right, there is a 'Filters' button (highlighted with a red box), a 'Default View' button, and a 'Columns' button. At the bottom right, there are 'Cancel' and 'Apply Filters' buttons. The footer features a 'HOME' button and the 'Magento U' logo.

Notes:

This slide shows the expanded filters feature. This interface opens when the "Filter" button is clicked.

Each filter is represented by its own input, which provides greater flexibility and the creation of more sophisticated filters.

Filters are covered in more detail later in this module.

3.7 Grids | layout.xml

Grids | layout.xml

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..">
    <update handle="styles"/>
    <body>
        <referenceBlock name="menu">
            <action method=" setActive">
                <argument name="itemId"
xsi:type="string">Magento_Catalog::catalog_products</argument>
            </action>
        </referenceBlock>
        <referenceBlock name="page.title">
            <action method="setTitleClass">
                <argument name="class" xsi:type="string">complex</argument>
            </action>
        </referenceBlock>
        <referenceContainer name="content">
            <uiComponent name="product_listing"/>
            <block class="Magento\Catalog\Block\Adminhtml\Product"
name="products_list"/>
        </referenceContainer>
    </body>
</page>
```

Grid Configuration

HOME

Magento U

Notes:

This code demonstrates how a grid is included into a page. As mentioned earlier, a grid is a UiComponent in Magento 2, so it should be enabled using the `<uiComponent>` layout directive.

3.8 Grids | Listing UiComponents



Notes:

This diagram presents a list of the various grid components:

Listing ... a central UiComponent that represents the grid itself.

Columns ... a component that includes a list of “Column” components. Each “Column” component represents a specific grid column and its data. As columns are now rendered in JavaScript, their functionality has been greatly expanded over Magento 1.

Filters ... a composite component that includes a set of “Filter” components. The diagram displays which native filter components are available in a native Magento installation.

Paging ... a component responsible for rendering paging.

Massactions ... a component that renders a massactions dropdown menu with a list of possible actions appropriate to a subset of a grid's records.

3.9 Grids | Listing UiComponent

Grids | Listing UiComponent

Listing UiComponent

HOME

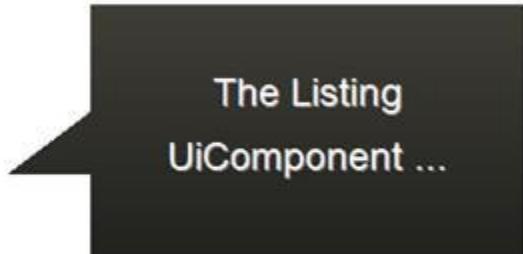
Magento U

Notes:

We'll now look at the Listing UiComponent in more detail.

3.10 ListingUiComponent | Overview

Listing UiComponent | Overview



- Is an analog of the Grid widget in Magento 1
- Has a group of children to facilitate its features
- Works with collections through the DataProvider

[HOME](#) **Magento U**

Notes:

The Listing UiComponent is:

- An analog of the Grid widget in Magento 1.
- Has a group of children to facilitate its features.
- Still works with collections (like Magento 1) through the DataProvider.

 **Note:** This may change in future releases.

We will look at each of these topics within this section.

3.11 Listing | Configuration (definitions.xml)

Listing UiComponent | Configuration (definition.xml)

```
<listing sorting="true" class="Magento\Ui\Component\Listing">
    <argument name="data" xsi:type="array">
        <item name="template" xsi:type="string">templates/listing/default</item>
        <item name="save_parameters_in_session" xsi:type="string">1</item>
        <item name="client_root" xsi:type="string">mui/index/render</item>
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">uiComponent</item>
        </item>
    </argument>
</listing>
```

UiComponent Class

Root Template

HOME

Magento U

Notes:

A code demonstration of the Listing configuration from a definition.xml file.

Note the UiComponent class -- Magento\Ui\Component\Listing, and the root template -- templates/listing/default.xhtml. This template is generic and is used like a container; all real rendering occurs by using HTML templates and JavaScript.

3.12 Listing UiComponent | Configuration (cms)

Listing UiComponent | Configuration (cms)

```
<argument name="data" xsi:type="array">
    <item name="js_config" xsi:type="array">
        <item name="provider" xsi:type="string">
            cms_page_listing.cms_page_listing_data_source</item>
        <item name="deps" xsi:type="string">
            cms_page_listing.cms_page_listing_data_source</item>
    </item>
    <item name="spinner" xsi:type="string">cms_page_columns</item>
    <item name="buttons" xsi:type="array">
        <item name="add" xsi:type="array">
            <item name="name" xsi:type="string">add</item>
            <item name="label" xsi:type="string" translate="true">Add New Page</item>
            <item name="class" xsi:type="string">primary</item>
            <item name="url" xsi:type="string">*//*new</item>
        </item>
    </item>
</argument>
```

JavaScript Configuration

Adding Buttons

HOME

Magento U

Notes:

In this code, you can see an example from the Cms module that demonstrates arguments a listing can use.

The first highlighted code shows the `js_config` argument -- it defines the configuration for JavaScript.

The second highlight shows a `buttons` argument, which specifies a list of buttons that can be added to the listing.

3.13 Grids | Filters



Notes:

Now, a further discussion of filters.

3.14 Filters | Overview

Filters | Overview

Filters in Magento 2 ...

- Are used to simplify searching through a list of records
- Can also set limits to display only those records that meet a set of criteria set by the Admin

HOME Magento U

Notes:

Filters are used to simplify searching through a list of records.

They can also set limits to display only those records that meet a set of criteria set by the Admin.

3.15 Grids | Filters Overview

The screenshot shows a 'Grids | Filters Overview' interface. At the top, there are three buttons: 'Filters' (highlighted in red), 'Default View', and 'Columns'. Below these are several filter fields grouped into sections: 'Name' (with 'Name' input), 'SKU' (with 'SKU' input), 'Type' (with 'Type' dropdown), 'Attribute Set' (with 'Attribute Set' dropdown), 'Visibility' (with 'Select...' dropdown), 'Status' (with 'Status' dropdown), 'Store View' (with 'Store View' dropdown), and 'Price' (with 'From' and 'To' inputs). There is also a 'Quantity' section with 'From' and 'To' inputs. At the bottom right are 'Cancel' and 'Apply Filters' buttons. The bottom navigation bar includes 'HOME' and the 'Magento U' logo.

Notes:

This slide again shows the expanded filters interface, first presented in the Grids overview.

Having the filters separated means you can now have more filters than visible fields, and the inputs for the filters can be complex combinations. For example, on the image you can see Name, SKU, and Type.

As filters are UiComponents, a list of all available filters can be found in `definition.xml`.

3.16 Filters | UiComponent

Filters | UiComponent

```
<filters class="Magento\Ui\Component\Filters">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">Magento_Ui/js/grid/filters/filters</item>
            <item name="displayArea" xsi:type="string">dataGridFilters</item>
            <item name="dataScope" xsi:type="string">filters</item>
            <item name="storageConfig" xsi:type="array">
                <item name="provider" xsi:type="string">ns = ${ $.ns },
                    index = bookmarks</item>
                <item name="namespace" xsi:type="string">current.filters</item>
            </item>
        </item>
        <item name="observers" xsi:type="array">
            <item name="column" xsi:type="string">column</item>
        </item>
    </argument>
</filters>
```

[HOME](#)

Magento U

Notes:

This code snippet shows the “filters” UiComponent configuration in default.xml.

Note the filters UiComponent and the set filter component; filters, act as a container for all specific filters.

Also on the slide you can see the JavaScript module, which renders this UiComponent.

3.17 Filters | Product Listing Example

Filters | Product Listing Example

```
class Filters extends \Magento\Ui\Component\Filters
{
    /**
     * @inheritDoc
     */
    public function update(UiComponentInterface $component)
    {
        if (!$component instanceof \Magento\Ui\Component\Filters) {
            return;
        }

        $attributeCodes = $component->getContext()
            ->getRequestParam('attributes_codes');
        if ($attributeCodes) {
            foreach ($this->getAttributes($attributeCodes) as $attribute) {
                $filter = $this->filterFactory->create($attribute,
                    $component->getContext());
                $filter->prepare();
                $component->addComponent($attribute->getAttributeCode(), $filter);
            }
        }
    }
}
```

[HOME](#)**Magento U**

Notes:

This slide presents an example of a Filters PHP class implementation for a products grid (`Magento\Catalog\Ui\Component\Listing\Filters`).

The code demonstrates how a list of filters is generated based on the product attributes.

3.18 Filters | Product Listing Example

Filters | Filter Pool

```
public function applyFilters(Collection $collection, SearchCriteriaInterface $criteria)
{
    foreach ($criteria->getFilterGroups() as $filterGroup) {
        foreach ($filterGroup->getFilters() as $filter) {
            /** @var $filterApplier FilterApplierInterface */
            if (isset($this->appliers[$filter->getConditionType()])) {
                $filterApplier = $this->appliers[$filter->getConditionType()];
            } else {
                $filterApplier = $this->appliers['regular'];
            }
            $filterApplier->apply($collection, $filter);
        }
    }
}
```

[HOME](#)**Magento U**

Notes:

How filters are applied depends upon the data provider implementation (which will be covered later).

The native data provider (`\Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider`) uses a `FilterPool` object (`\Magento\Framework\View\Element\UiComponent\DataProvider\FilterPool`), which contains an `applyFilters` method (shown on the slide) that applies filters.

Configuration of the filter appliers seen on the slide can be defined in the `di.xml` for a specific grid.

3.19 Filters | Filter Pool

Filters | Filter Pool

```
class FilterPool
{
    /**
     * @var array
     */
    protected $filters = [];

    /**
     * @var array
     */
    protected $appliers;

    /**
     * @param array $appliers
     */
    public function __construct(array $appliers = [])
    {
        $this->appliers = $appliers;
    }
}
```

Appliers Injected Here

[HOME](#)

Magento U

Notes:

As you can see from this second code snippet, the process goes through a list of registered appliers, which it uses to apply a filter.

In general, the list of appliers is injected using DI.

3.20 Reinforcement Exercise (6.3.1)

Reinforcement Exercise (6.3.1): Configurable Products

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

3.21 Grids | DataSource

Grids | DataSource

DataSource

HOME

Magento U

A slide with an orange header and footer. The header contains the text "Grids | DataSource". The footer contains a "HOME" button and the "Magento U" logo. The main content area features a large orange graphic on the left and the word "DataSource" in the center.

Notes:

The next topic we'll cover is DataSource.

3.22 DataSource | Overview

DataSource | Overview

DataSource ...

HOME

Magento U

Notes:

DataSource is a UiComponent that provides data to other components.

3.23 DataSource | Default DataSource

DataSource | Default DataSource

- Does not actually contain data.
- Works with DataProvider class to obtain data.
- **DataSource and DataProvider are usually re-declared on each specific grid.**

[HOME](#)**Magento U****Notes:**

As discussed earlier, DataSource requires DataProvider. DataProvider is actually the class which obtains data. It has to implement a specific interface to provide JSON data to DataSource.

You usually implement your own DataProvider for an instance of a UiComponent.

We will learn more about DataProviders on the slides that follow.

3.24 DataSource | Default DataSource Configuration

DataSource | Default DataSource Configuration

```
<dataSource class="Magento\Ui\Component\DataSource"/>

<listing sorting="true" class="Magento\Ui\Component\Listing">
  ...
</listing>
```

[HOME](#)

Magento U

Notes:

There are a few commonly used UiComponents available in Magento 2. Two of them -- listing and form -- are composite and contain a set of child components.

DataSource is also a UiComponent that provides data to all of them.

Of course, data needs to be taken from the database. But DataSource is a JavaScript module, so it uses DataProvider , which is a PHP class, to generate data for DataSource.

The code on this slide demonstrates configuration for the DataSource component in the `definition.xml` file.

3.25 DataSource | Default DataSource

DataSource | Abstract DataProvider

Class: Magento\Ui\DataProvider\AbstractDataProvider

- Wraps collection's operations like sorting, filtering in the current implementation.
- Implements `getData()` method, which is used to extract data to JavaScript; has to return an array.
- Has abstract method `getCollection()`, which needs to be implemented in a specific DataProvider.

```
public function getData()
{
    return $this->getCollection()->toArray();
}
```

HOME

Magento U

Notes:

AbstractDataProvider is a class that every DataProvider has to extend; its key method `getData` has to be implemented.

On the slide, you can see an example of this type of implementation. The `getData` method returns data from a collection.

This is typical of grids -- their DataProvider classes usually only wrap collection classes, which extract data from the database.

-  **Note:** There is another abstract data provider available:
`\Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider`. Those classes are used in different places in the core code.

3.26 DataSource | Catalog Configuration

DataSource | Catalog Configuration

```
<dataSource name="product_listing_data_source">
    <argument name="dataProvider" xsi:type="configurableObject">
        <argument name="class" xsi:type="string">Magento\Catalog\Ui\DataProvider\Product\ProductDataProvider</argument>
        <argument name="name" xsi:type="string">product_listing_data_source
        </argument>
        <argument name="primaryFieldName" xsi:type="string">entity_id
        </argument>
        <argument name="requestFieldName" xsi:type="string">id</argument>
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="update_url" xsi:type="url"
                    path="mui/index/render"/>
            </item>
        </argument>
    </argument>
</dataSource>
```

Listing-specific Data Provider

[HOME](#) **Magento U**

Notes:

Here is an example of the DataSource component configuration for product listing.

You can see that it defines the DataProvider class. Usually, the DataProvider class is defined for each implementation of a UiComponent.

3.27 DataSource | Catalog DataProvider

DataSource | Catalog DataProvider

```
public function getData()
{
    if (!$this->getCollection()->isLoaded()) {
        $this->getCollection()->load();
    }
    $items = $this->getCollection()->toArray();

    return [
        'totalRecords' => $this->getCollection()->getSize(),
        'items' => array_values($items),
    ];
}
public function addField($field, $alias = null)
{
    if (isset($this->addFieldStrategies[$field])) {
        $this->addFieldStrategies[$field]->addField($this->getCollection(), $field, $alias);
    } else {
        parent::addField($field, $alias);
    }
}
public function addFilter($condition, $field = null, $type = 'regular')
{
    if (isset($this->addFilterStrategies[$field])) {
        $this->addFilterStrategies[$field]->addFilter($this->getCollection(), $field, $condition);
    } else {
        parent::addFilter($condition, $field);
    }
}
```

[HOME](#)**Magento U**

Notes:

This slide shows an implementation of DataProvider for the product_listing UiComponent. As you can see, getData loads a collection and creates an array with “totalRecords” and “items” elements.

You can also see a couple of other methods in the code; for example, addFilter(). It is just a wrapper around collection filtering.

The DataSource Catalog DataProvider implements three methods:

- `getData()`
- `addField()`
- `addFilter()`

3.28 Reinforcement Exercise (6.3.2)

Reinforcement Exercise (6.3.2): Add a New Attribute

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

[HOME](#)

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

4. Grids Part 2

4.1 Grids: Part 2

Grids: Part 2



The illustration shows a computer monitor with an orange frame. On the screen, there is a window with a header containing a gear icon and a progress bar. Below the header is a grid of six orange squares arranged in two rows of three. A red cursor arrow points to the bottom-right square of the grid. The monitor sits on a dark brown stand.

HOME

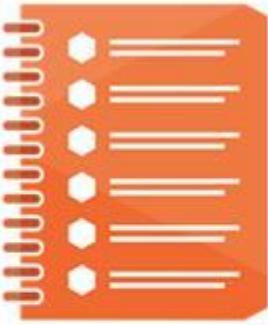
Magento U

Notes:

We now continue our discussion of grids.

4.2 Module Topics | Grids

Module Topics | Grids



In this module, we will discuss...

- UiComponents
- Filters
- DataSource
- Grid indexer
- Columns
- Paging

[HOME](#) **Magento U**

Notes:

In this module, we will focus on listing UiComponents, filters, DataSource, the grid indexer, columns, and paging.

4.3 Grids | Grid Indexer

Grids | Grid Indexer

Grid Indexer

HOME

Magento U

4.4 Grid Indexer | Overview

Grid Indexer | Overview

The grid indexer refers to...

- The Magento 1 sales module has a special set of tables designed for admin grids. Tables are populated on-the-fly when an entity is saved.
- A similar approach has been expanded in Magento 2 to include more entities.

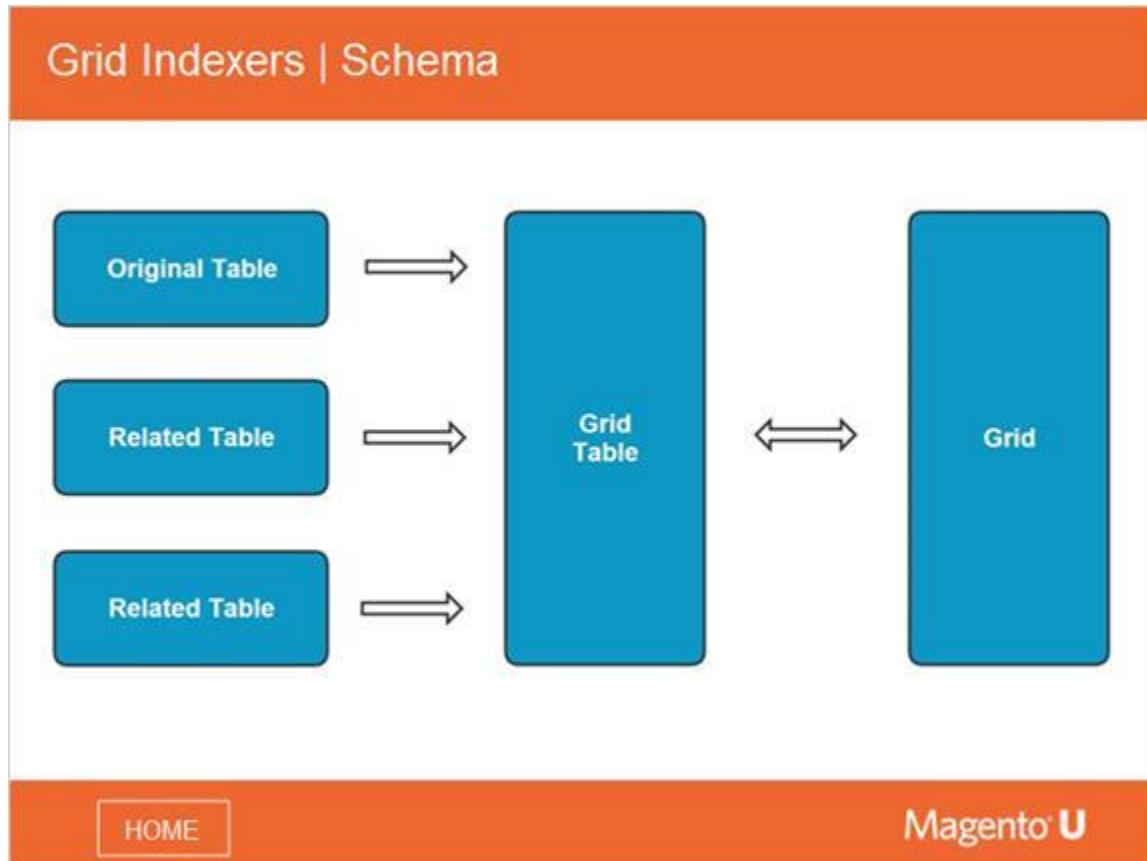
[HOME](#) [Magento U](#)

Notes:

In Magento 1, the sales module contains a special set of tables designed for admin grids. These tables populate on the fly, when an entity is saved.

A similar approach is used in Magento 2, but it has been expanded to include more entities.

4.5 Grid Indexers | Schema



Notes:

Depicted on the slide is a typical schema for grid indexing. Grid indexing solves the problem of joining multiple tables (usually one main table and sets of related ones) in Magento 2.

However, this can result in a large query that doesn't perform well.

A solution is to populate a special "grid" table with the required data at the time of entity saving. This is the same approach used in Magento 1 for sales entities.

4.6 Grid Indexer | Orders Example 1

Grid Indexers | Orders Example 1

```
<dataSource name="sales_order_grid_data_source">
    <argument name="dataProvider" xsi:type="configurableObject">
        <argument name="class" xsi:type="string">
            Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider
        </argument>
        <argument name="name" xsi:type="string">
            sales_order_grid_data_source</argument>
        <argument name="primaryFieldName" xsi:type="string">entity_id</argument>
        <argument name="requestFieldName" xsi:type="string">id</argument>
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="update_url" xsi:type="url" path="mui/index/render"/>
                <item name="component" xsi:type="string">
                    Magento_Ui/js/grid/provider</item>
            </item>
        </argument>
    </argument>
</dataSource>
```

[HOME](#)**Magento U**

Notes:

This is a listing configuration for a sales order. As you can see, it uses an abstract data provider (from the framework, not from the Magento_Ui module), which is configured in the `di.xml` (as shown on the next slide).

4.7 Grid Indexer | Orders Example 2

Grid Indexers | Orders Example 2

```
<type name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory">
<arguments>
<argument name="collections" xsi:type="array">
<item name="sales_order_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Grid\Collection</item>
<item name="sales_order_invoice_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Invoice\Grid\Collection</item>
<item name="sales_order_shipment_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Shipment\Grid\Collection</item>
<item name="sales_order_creditmemo_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Creditmemo\Grid\Collection</item>
<item name="sales_order_view_invoice_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Invoice\Orders\Grid\Collection</item>
<item name="sales_order_view_shipment_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Shipment\Order\Grid\Collection</item>
<item name="sales_order_view_creditmemo_grid_data_source" xsi:type="string">
    Magento\Sales\Model\ResourceModel\Order\Creditmemo\Order\Grid\Collection</item>
</argument>
</arguments>
</type>
```

[HOME](#)**Magento U**

Notes:

This configuration from the sales module `di.xml` file shows how grid tables are implemented for sales entities. As you can see, it attaches special collection classes to data sources related to the sales entities.

4.8 Grid Indexer | Orders Example 3

```
class Collection extends \Magento\Framework\View\Element\UiComponent\DataProvider\SearchResult
{
    /**
     * Initialize dependencies.
     *
     * @param EntityFactory $entityFactory
     * @param Logger $logger
     * @param FetchStrategy $fetchStrategy
     * @param EventManager $eventManager
     * @param string $mainTable
     * @param string $resourceModel
     */
    public function __construct(
        EntityFactory $entityFactory,
        Logger $logger,
        FetchStrategy $fetchStrategy,
        EventManager $eventManager,
        $mainTable = 'sales_order_grid',
        $resourceModel = '\Magento\Sales\Model\ResourceModel\Order'
    ) {
        parent::__construct($entityFactory, $logger, $fetchStrategy, $eventManager, $mainTable,
            $resourceModel);
    }
}
```

[HOME](#)**Magento U****Notes:**

And finally, looking at a grid collection, you can see it takes data from a special table: `sales_order_grid`.

4.9 Reinforcement Exercise (6.4.1)

Reinforcement Exercise (6.4.1): Add a New Column

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

4.10 Grids | Columns

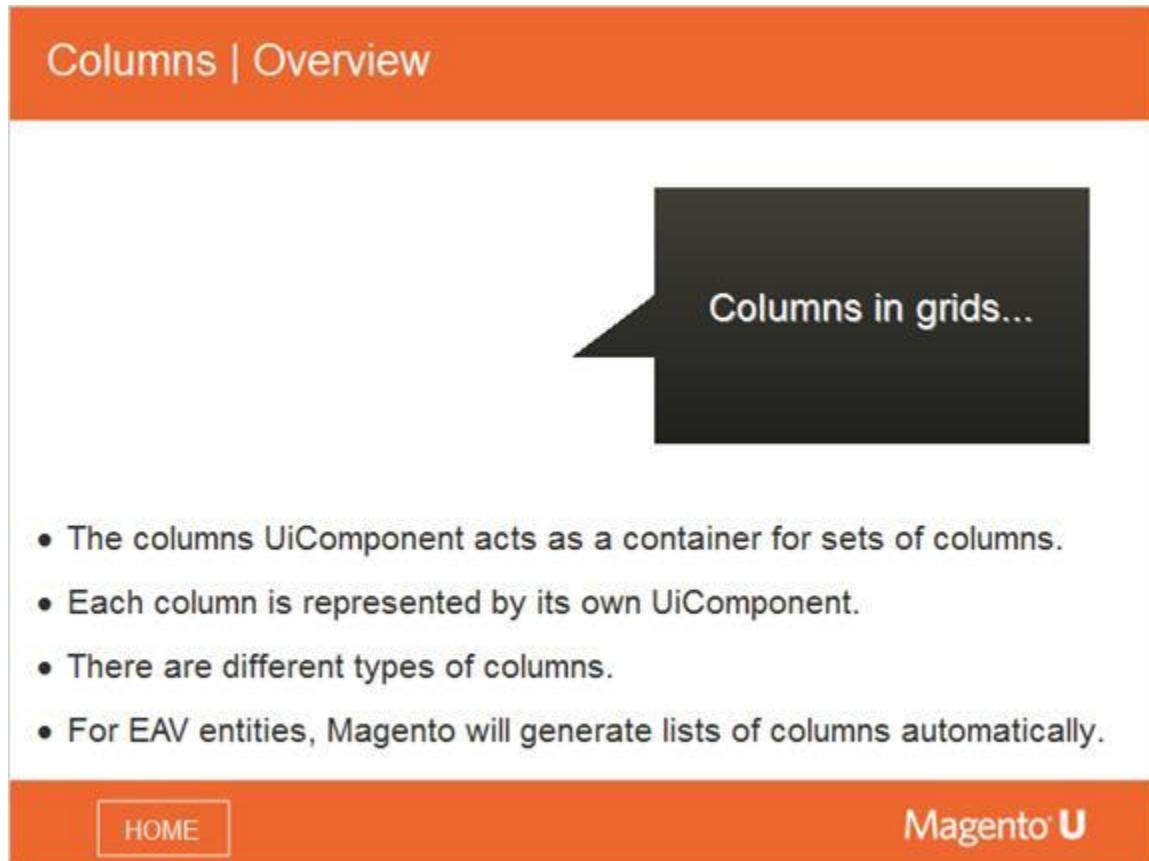
Grids | Columns

Columns

HOME

Magento U

4.11 Columns | Overview



Columns | Overview

Columns in grids...

- The columns UiComponent acts as a container for sets of columns.
- Each column is represented by its own UiComponent.
- There are different types of columns.
- For EAV entities, Magento will generate lists of columns automatically.

HOME Magento U

Notes:

For rendering, the columns listing component uses the same approach as with filters.

There is a Columns UiComponent that acts as a container, and a set of specific Column UiComponents which render specific columns.

In Magento 1, you have to configure columns in the `_prepareColumns()` method of a grid object. In Magento 2, each column is a UiComponent, so it is configured in the component's XML file.

It is important that EAV entities obtain information about columns automatically, although you can see column configuration in the `product_listing.xml` file.

4.12 Columns | definitions.xml

Columns | definition.xml

```
<columns class="Magento\Ui\Component\Listing\Columns">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">
                Magento_Ui/js/grid/listing</item>
            <item name="componentType" xsi:type="string">columns</item>
            <item name="storageConfig" xsi:type="array">
                <item name="provider" xsi:type="string">
                    ns = ${ $.ns }, index = bookmarks</item>
                <item name="namespace" xsi:type="string">current</item>
            </item>
            ...
        </item>
    </argument>
</columns>

<column class="Magento\Ui\Component\Listing\Columns\Column">
    <argument name="data" xsi:type="array">
        <item name="state_prefix" xsi:type="string">columns</item>
        <item name="config" xsi:type="array">
            <item name="component"
xsi:type="string">Magento_Ui/js/grid/columns/column</item>
            <item name="componentType" xsi:type="string">column</item>
            <item name="dataType" xsi:type="string">text</item>
        </item>
    </argument>
</column>
```

[HOME](#)[Magento U](#)

Notes:

The columns UiComponent is defined in the definition.xml file.

In the above code, you can see the columns component with the corresponding class `Magento\Ui\Component\Listing\Columns` (first highlighted code) and the column component with the class `Magento\Ui\Component\Listing\Columns\Column` (second highlight).

4.13 Columns | Column JavaScript

Columns | Column JavaScript

Magento/Ui/view/base/web/js/grid/columns/*

```
graph TD; Root["Magento/Ui/view/base/web/js/grid/columns/*"] --> columnjs["column.js"]; Root --> datejs["date.js"]; Root --> multiselectjs["multiselect.js"]; Root --> actionsjs["actions.js"]; Root --> thumbnailjs["thumbnail.js"]; Root --> onoffjs["onoff.js"]
```

HOME

Magento U

Notes:

This slide lists JavaScript modules that are responsible for rendering columns.

4.14 Columns | column.js

Columns | column.js

- Basic JavaScript module for column.
- Looking in its constructor, you can see which options are configurable in an xml file.

HOME Magento U

Notes:

column.js is a JavaScript module that renders a column. If you open the file to take a look at the constructor, you will see the following code.

This code provides information about which configuration options are available in the listing xml files:

```
return Element.extend({
    defaults: {
        headerTpl: 'ui/grid/columns/text',
        bodyTpl: 'ui/grid/cells/text',
        disableAction: false,
        controlVisibility: true,
        sortable: true,
        sorting: false,
        visible: true,
        draggable: true,
        fieldClass: {},
        ignoreTmpls: {
            fieldAction: true
        },
        statefull: {
            visible: true,
        }
});
```

```
        sorting: true
    },
    imports: {
        exportSorting: 'sorting'
    },
    listens: {
        '${ $.provider }:params.sorting.field': 'onSortChange'
    },
    modules: {
        source: '${ $.provider }'
    }
}
```

4.15 Columns | definitions.xml

Columns | Typical Column Configuration

```
<column name="is_active">
    <argument name="data" xsi:type="array">
        <item name="options" xsi:type="object">
            Magento\Cms\Model\Page\Source\IsActive</item>
        <item name="config" xsi:type="array">
            <item name="filter" xsi:type="string">select</item>
            <item name="component" xsi:type="string">
                Magento_Ui/js/grid/columns/select</item>
            <item name="editor" xsi:type="string">select</item>
            <item name="dataType" xsi:type="string">select</item>
            <item name="label" xsi:type="string" translate="true">Status</item>
        </item>
    </argument>
</column>
```

[HOME](#)

Magento U

Notes:

This code displays an example of column configuration in the Cms module (Cms Page grid).

This column is of the type `select`, so you can specify options. The options are set in the `Magento\Cms\Model\Page\Source\IsActive` class.

You can also see that the component that renders this column is `select.js`.

4.16 Columns | Invisible Columns

The screenshot shows the Magento 2 Admin Panel with a search bar at the top. Below it is a table with columns: ID, Title, URL Key, Layout, Store View, and Status. There are 6 records found. A 'Columns' button is visible above the table. A modal window titled 'Columns' is open, showing two tabs: '15 out of 15 visible' and '9 out of 15 visible'. Both tabs list various columns with checkboxes. The '15 out of 15 visible' tab has checkboxes for ID, Title, URL Key, Layout, Store View, Status, Created, Modified, Custom design from, Custom theme, Custom layout, Meta Keywords, Meta Description, and Action. The '9 out of 15 visible' tab has checkboxes for Title, URL Key, Layout, Store View, Status, Created, Modified, Custom design from, Custom theme, and Action. At the bottom of the modal are 'Reset' and 'Cancel' buttons. In the bottom right corner of the main interface, there is a 'HOME' button and the 'Magento U' logo.

Notes:

Magento 2 offers a unique feature not available in Magento 1: dynamic columns. This is possible because of the flexibility that JavaScript rendering provides.

As you can see on the slide, there is a “Columns” button that opens a list of all the available columns. Columns with a check mark are those currently rendered; the remaining columns are those that can be rendered later.

There is also an option in the column configuration (Boolean visible) that defines whether a column is rendered by default or not. The next slide illustrates this.

4.17 Columns | Invisible Column Configuration

Columns | Visible-Invisible Column Configuration

```
<column name="custom_theme">
    <argument name="data" xsi:type="array">
        <item name="options" xsi:type="object">
            Magento\Cms\Model\Page\Source\Theme</item>
        <item name="config" xsi:type="array">
            <item name="filter" xsi:type="string">select</item>
            <item name="component" xsi:type="string">
                Magento_Ui/js/grid/columns/select</item>
            <item name="editor" xsi:type="string">select</item>
            <item name="dataType" xsi:type="string">select</item>
            <item name="label" xsi:type="string" translate="true">Custom Theme
            </item>
            <item name="visible" xsi:type="boolean">false</item>
        </item>
    </argument>
</column>
```

[HOME](#)**Magento U****Notes:**

Here is the code example for a non-visible column configuration.

4.18 Reinforcement Exercise (6.4.2)

Reinforcement Exercise (6.4.2): Add a New Attribute

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

4.19 Grids | Mass Actions

Grids | Mass Actions

Mass Actions

HOME

Magento U

Notes:

We now move to a discussion of mass actions.

4.20 Mass Actions | Overview

Mass Actions | Overview

Mass actions...

HOME

Magento U

Notes:

Mass actions are used to apply certain actions (events) to a set of records.

They define which actions are available on the grid page, and list actions configured in the `data/config/actions` argument.

4.21 Mass Actions | Area Location Diagram

The screenshot shows the Magento Catalog grid interface. At the top, there's an orange header bar with the title "Mass Actions | Area Location Diagram". Below it is a white header with "Catalog" and a search bar. A red "Add Product" button is visible in the top right. The main area contains a grid of product items, each with a "Select" checkbox and a small thumbnail image. The columns include "Thumbnail", "Name", "Type", "SKU/Media ID", "Title", "Price", "Quantity", "Visibility", "Status", "Attributes", and "Actions". A dropdown menu labeled "Actions" is open over the first few rows. The bottom of the grid has a toolbar with buttons for "200", "per page", and navigation arrows. At the very bottom, there's an orange footer bar with a "HOME" button on the left and the "Magento U" logo on the right.

Thumbnail	Name	Type	SKU/Media ID	Title	Price	Quantity	Visibility	Status	Attributes	Actions
	Black Duffle Bag	Simple Product	Bag	2040001	\$20.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Black Shoulder Pack	Simple Product	Bag	2040002	\$30.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Cross Summit Backpack	Simple Product	Bag	2040003	\$35.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Wayfare Messenger Bag	Simple Product	Bag	2040004	\$45.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Black Field Messenger	Simple Product	Bag	2040005	\$45.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Fusion Backpack	Simple Product	Bag	2040006	\$50.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Impulse Duffle	Simple Product	Bag	2040007	\$75.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Vintage Logo Bag	Simple Product	Bag	2040008	\$80.00	100.0000	Catalog, Search	Enabled	Main Website	Edit
	Computer Track Tote	Simple Product	Bag	2040009	\$82.00	100.0000	Catalog, Search	Enabled	Main Website	Edit

Notes:

Recall from the general Area Location diagram where the Massactions block is situated.

The massactions dropdown can optionally be made available on a grid page. If so, it defines a list of actions that can be applied to a select set of records from the grid.

Conceptually, massactions is a UiComponent, child of a Listing.

4.22 Mass Actions | definitions.xml

Mass Actions | definition.xml

```
<massaction class="Magento\Ui\Component\MassAction">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">
                Magento_Ui/js/grid/massactions</item>
            <item name="displayArea" xsi:type="string">bottom</item>
        </item>
    </argument>
</massaction>
```

[HOME](#)**Magento U****Notes:**

In this code example, you can see the generic configuration for massactions defined in `definition.xml`.

In that file, you can see the corresponding `UiComponent` class - `Magento\Ui\Component\MassAction`, the template - `templates/listingcontainer/massaction/default.xhtml`, and the JavaScript module responsible for rendering - `massactions.js`.

4.23 Mass Actions | Actions Config Example

Mass Actions | Action Config Example

```
<massaction name="listing_massaction">
    <action name="delete">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="type" xsi:type="string">delete</item>
                <item name="label" xsi:type="string" translate="true">
                    Delete</item>
                <item name="url" xsi:type="url" path="cms/page/massDelete"/>
                <item name="confirm" xsi:type="array">
                    <item name="title" xsi:type="string" translate="true">
                        Delete items</item>
                    <item name="message" xsi:type="string" translate="true">
                        Are you sure you want to delete selected items?</item>
                </item>
            </item>
        </argument>
    </action>
    <action name="disable">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="type" xsi:type="string">disable</item>
                <item name="label" xsi:type="string" translate="true">
                    Disable</item>
                <item name="url" xsi:type="url" path="cms/page/massDisable"/>
            </item>
        </argument>
    </action>

```

[HOME](#)**Magento U****Notes:**

Here is the configuration from a Cms page grid. You see the possible actions -- Delete, Disable -- along with the corresponding URLs to send data, and possible confirmation messages.

4.24 Mass Actions | JavaScript Module

Mass Actions | JavaScript Module

`Magento/Ui/view/base/web/js/grid/massactions.js`

HOME Magento U

Notes:

The JavaScript module responsible for `massactions` rendering is `massactions.js`.

4.25 Reinforcement Exercise (6.4.3)

Reinforcement Exercise (6.4.3): Mass Action

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

4.26 Grids | Paging

Grids | Paging

Paging

HOME

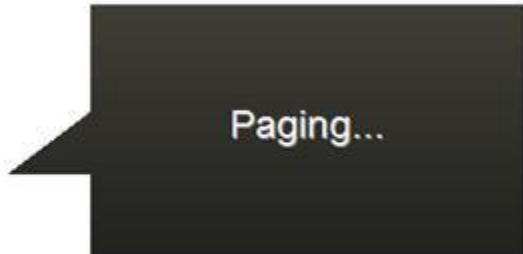
Magento U

Notes:

Now, we'll discuss the final topic in this module, Paging.

4.27 Paging | Overview

Paging | Overview



The paging component is used to render paging elements:
next / previous pages, current page, total number of
records, set of closest next / previous page numbers.

[HOME](#) **Magento U**

Notes:

The paging component is used to render paging elements:

- Next / previous pages
- Current page
- Total number of records
- Set of closest next / previous page numbers

4.28 Paging | Overview Diagram

The screenshot shows the Magento 2 Admin Catalog grid. At the top, there's a search bar and a red 'Add Product' button. Below the header, a message '2048 records found' is displayed. The main area contains a table with product data, including columns for ID, Product ID, Name, Type, Status, Price, Quantity, Visibility, Status, Websites, and Actions. A 'paging' section is overlaid on the grid, containing controls for 'per page' (set to 20), page navigation ('1 of 1'), and a 'refresh' button. At the bottom left is a 'HOME' button, and at the bottom right is the 'Magento U' logo.

ID	Product ID	Name	Type	Status	Price	Quantity	Visibility	Status	Websites	Actions
1	20480001	Black Duffle Bag	Simple Product	Enabled	\$60.00	100.0000	Catalog, Search	Enabled	Main Website	
2	20480002	Blue Shoulder Pack	Simple Product	Enabled	\$45.00	100.0000	Catalog, Search	Enabled	Main Website	
3	20480003	Cross-Body Backpack	Simple Product	Enabled	\$55.00	100.0000	Catalog, Search	Enabled	Main Website	
4	20480004	Keychain Messenger Bag	Simple Product	Enabled	\$40.00	100.0000	Catalog, Search	Enabled	Main Website	
5	20480005	Red Field Messenger	Simple Product	Enabled	\$45.00	100.0000	Catalog, Search	Enabled	Main Website	
6	20480006	Yellow Backpack	Simple Product	Enabled	\$60.00	100.0000	Catalog, Search	Enabled	Main Website	
7	20480007	Orange Duffle	Simple Product	Enabled	\$50.00	100.0000	Catalog, Search	Enabled	Main Website	
8	20480008	Yellow Logo Bag	Simple Product	Enabled	\$55.00	100.0000	Catalog, Search	Enabled	Main Website	
9	20480009	Computer Track Case	Simple Product	Enabled	\$62.00	100.0000	Catalog, Search	Enabled	Main Website	

Notes:

Again using the general Area Location diagram, you can see where the Paging feature is situated.

Paging is also a UiComponent. Its main responsibility is to display paging control options, like number of records per page, page number, and page navigation.

4.29 Paging | default.xml

Paging | default.xml

```
<paging class="Magento\Ui\Component\Paging">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">
                Magento_Ui/js/grid/paging/paging</item>
            <item name="displayArea" xsi:type="string">bottom</item>
            <item name="storageConfig" xsi:type="array">
                <item name="provider" xsi:type="string">
                    ns = ${ $.ns }, index = bookmarks</item>
                <item name="namespace" xsi:type="string">current.paging</item>
            </item>
        </item>
    </argument>
</paging>
```

[HOME](#)**Magento U****Notes:**

The code example shows paging configuration in the file, `definition.xml`. As you can see, it is quite simple.

Note that the JavaScript module is `paging.js`, located under the `Magento/Ui` module.

4.30 Paging | Example

Paging | Example

```
<listingToolbar>
  -
  <paging name="listing_paging" />
</listingToolbar>
```

[HOME](#)**Magento U****Notes:**

This slide shows an example of paging configuration for the product grid.

4.31 Paging | Default Options

Paging | Default Options

```
namespace Magento\Ui\Component;
class Paging extends AbstractComponent
{
    const NAME = 'paging';
    protected $_data = [
        'config' => [
            'options' => [
                '20' => [
                    'value' => 20,
                    'label' => 20
                ],
                ...
                '100' => [
                    'value' => 100,
                    'label' => 100
                ],
                '200' => [
                    'value' => 200,
                    'label' => 200
                ],
                ...
                'pageSize' => 20,
                'current' => 1
            ]
        ];
}
```

[HOME](#)

Magento U

Notes:

Here you can see how the default paging options are configured in its PHP UiComponent class.

4.32 Reinforcement Exercise (6.4.4)

Reinforcement Exercise (6.4.4): Create a Table

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

5. Forms

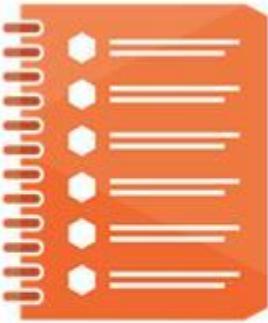
5.1 Adminhtml Forms



Notes:

5.2 Module Topics | Adminhtml Forms

Module Topics | Adminhtml Forms



In this module, we will discuss...

- Forms overview
- Form components
- Fieldsets
- Form elements

[HOME](#) **Magento U**

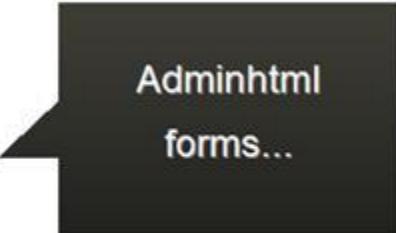
Notes:

In this module, we will discuss:

- Forms overview
- Form components
- Fieldsets
- Form elements

5.3 Adminhtml Forms | Definition

Adminhtml Forms | Definition



- A form in Magento 2 is a framework which creates effective user interfaces for managing different entities.
- Almost all backend interfaces are built from forms and grids.

[HOME](#) **Magento U**

Notes:

A form in Magento 2 is a framework, which creates effective user interfaces for managing different entities.

Almost all backend interfaces are built from forms and grids.

5.4 Adminhtml Forms | Format & Fields

The screenshot shows the 'Format & Fields' configuration page in the Magento Adminhtml interface. The page has an orange header bar with the title 'Adminhtml Forms | Format & Fields'. Below the header is a large grey form area containing various input fields and dropdown menus. The fields include:

- Associate to Website: Main Website (dropdown)
- Group: General (dropdown)
- Disable Automatic Group Change Based on VAT ID: (checkbox)
- Prefix: (text input)
- First Name: Veronica (text input)
- Middle Name/Initial: (text input)
- Last Name: Costello (text input)
- Suffix: (text input)
- Email: @email: roni_cost@dmall.com (text input)
- Date Of Birth: (date input)
- Tax/VAT Number: (text input)
- Gender: Male (dropdown)
- Send From: Default Store View (dropdown)
- Welcome Email: (checkbox) Send a Welcome email

At the bottom left is a 'HOME' button, and at the bottom right is the 'Magento U' logo.

Notes:

Here is an example of the form backend interface.

5.5 Adminhtml Forms | Form Container

The screenshot shows a screenshot of the Magento Adminhtml Forms | Form Container page. At the top, there is a header bar with the title "Adminhtml Forms | Form Container". Below the header is a navigation bar with links: "Back", "Delete Customer", "Reset", "Create Order", "Reset Password", "Force Sign-In", and "Save and Continue Edit". The main content area is titled "Account Information". It contains several input fields and dropdown menus: "Associate to Website" (set to "Main Website"), "Group" (set to "General"), and a checkbox for "Disable Automatic Group Change Based on VAT ID". There is also a "Prefix" input field. At the bottom right of the content area is a red "Save Customer" button. The footer of the page includes a "HOME" link and the "Magento U" logo.

Notes:

We will start by looking at a form container. Its main goal is to display actions applicable to the whole form -- for example, Save.

5.6 Adminhtml Forms | Magento 2 vs. Magento 1 Approaches

Adminhtml Forms | Magento 2 vs. Magento 1 Approaches

There are 2 approaches for rendering forms in Magento 2:

- Using UiComponent
- Using PHP Form objects (as in Magento 1)

[HOME](#)

Magento U

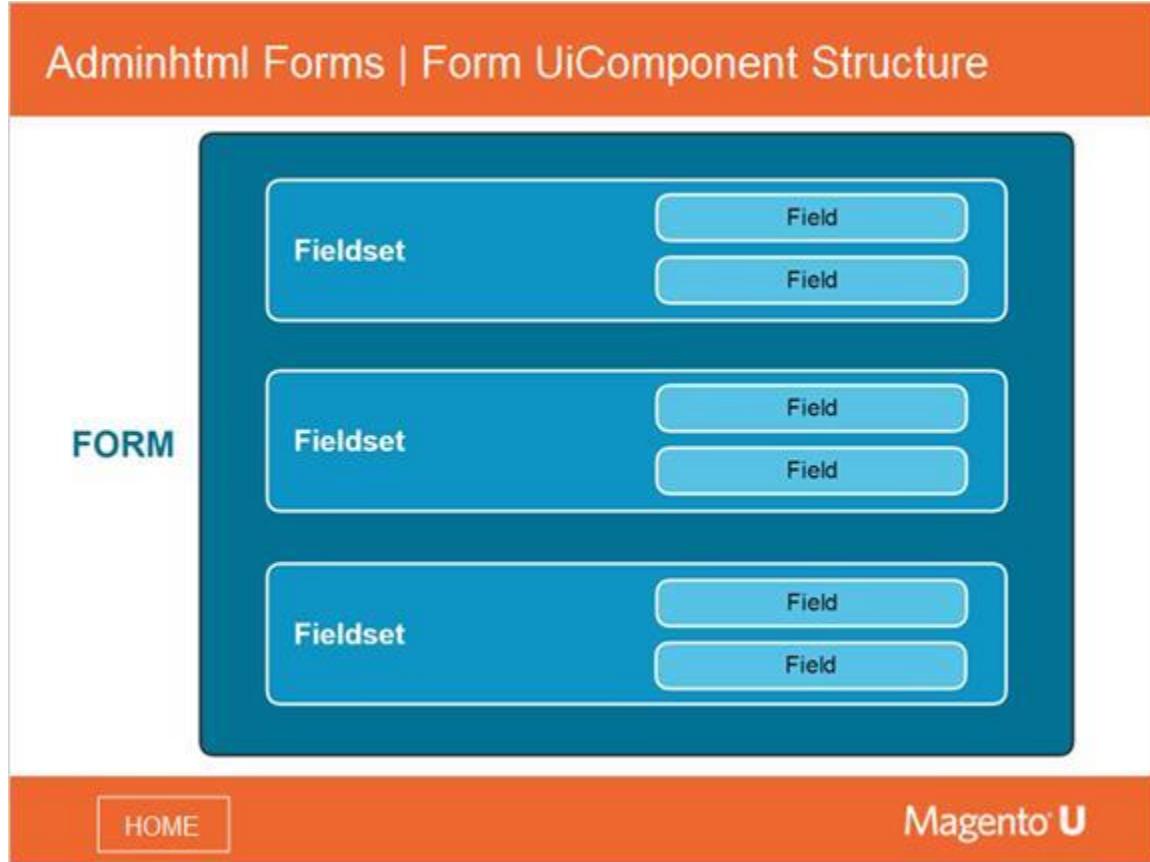
Notes:

In Magento 1, forms are generated using a standard layout mechanism.

In Magento 2, there are two options: using a set of PHP classes (blocks), or the UiComponent.

In this course, we will cover the UiComponent approach.

5.7 Adminhtml Forms | Form UiComponent Structure



Notes:

The Form UiComponent's PHP class is: `Magento\Ui\Component\Form`

Its most important function is to obtain data:

```
public function getDataSourceData()
{
    $dataSource = [];

    $id = $this->getContext()->getRequestParam($this->getContext()->getDataProvider()->getRequestFieldName(), null);
    $filter = $this->filterBuilder->setField($this->getContext()->getDataProvider()->getPrimaryFieldName())
        ->setValue($id)
        ->create();
    $this->getContext()->getDataProvider()
        ->addFilter($filter);

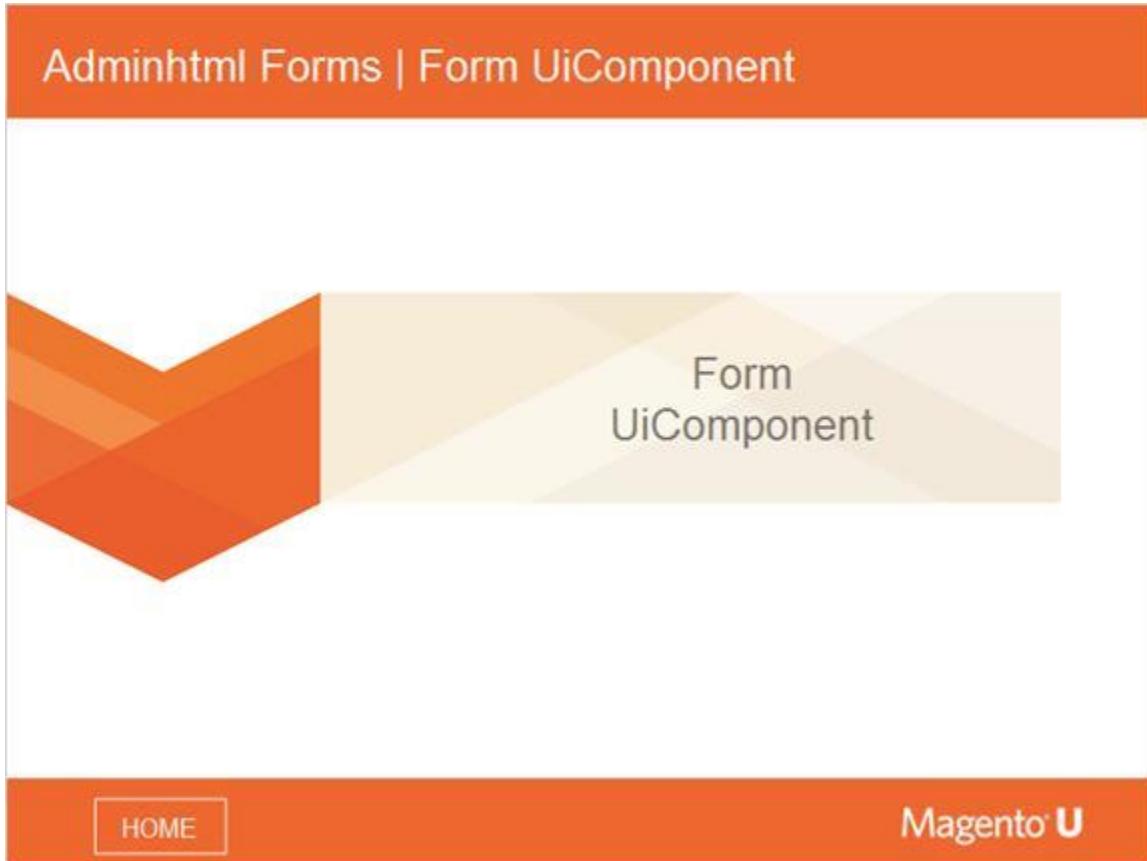
    $data = $this->getContext()->getDataProvider()->getData();

    if (isset($data[$id])) {
        $dataSource = [
            'label' => $this->__(ucwords($this->getLabel($id)))
        ];
    }
}
```

```
        'data' => $data[$id]
    ];
} elseif (isset($data['items'])) {
    foreach ($data['items'] as $item) {
        if ($item[$item['id_field_name']] == $id) {
            $dataSource = ['data' => ['general' => $item]];
        }
    }
}
return $dataSource;
}
```

As you can see, it checks whether the “id” parameter is set, and if so, adds a primary key filter.

5.8 Adminhtml Forms | Form UiComponent



5.9 Form UiComponent | Code Definition

Form UiComponent | Code Definition

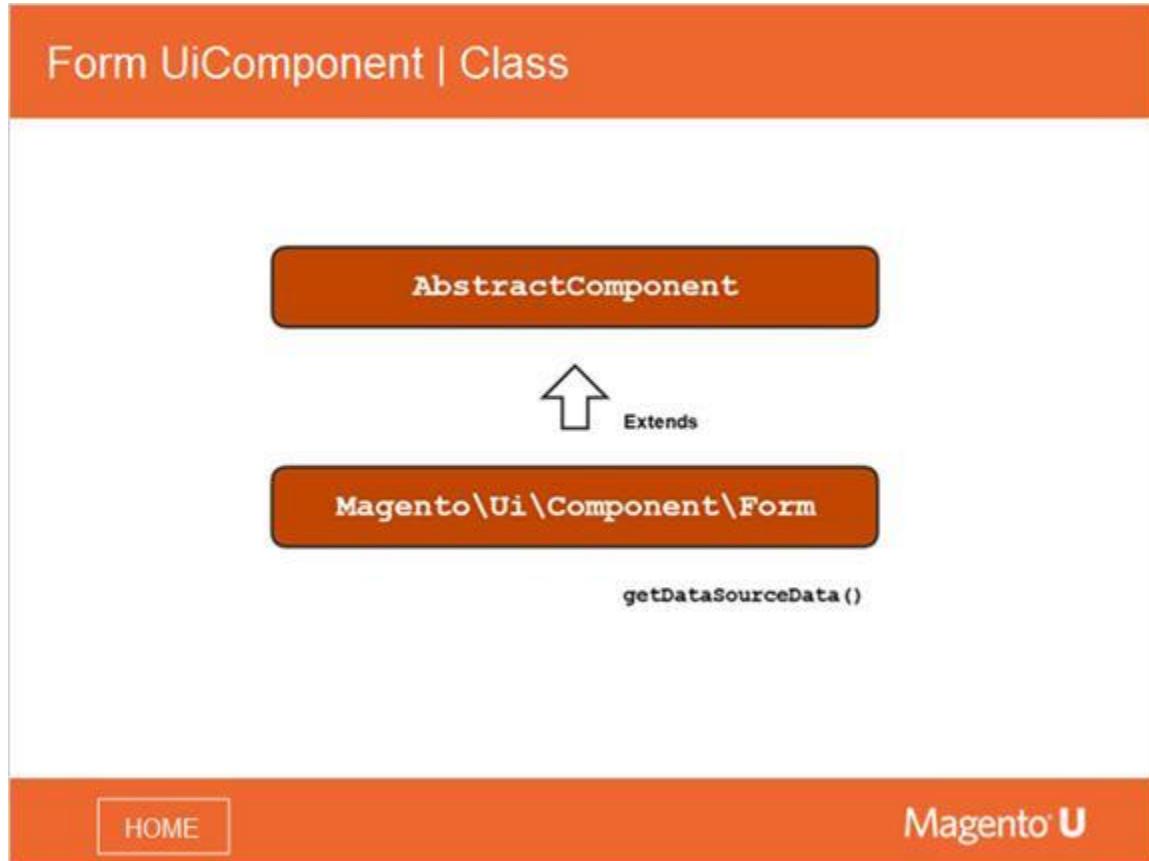
```
<form class="Magento\Ui\Component\Form">
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="component" xsi:type="string">Magento_Ui/js/form/form</item>
        </item>
        <item name="template" xsi:type="string">templates/form/default</item>
    </argument>
</form>
```

[HOME](#)**Magento U****Notes:**

This code shows how the Form component is defined in the file: `definition.xml`.

From this XML code, you can see that the component's code is located in `Magento/Ui/view/base/web/js/form/form.js`, and the default template in: `Magento/Ui/view/base/ui_component/templates/form/default.xhtml`

5.10 Form UiComponent | DataProvider Customer Example



Notes:

The Form UiComponent's PHP class is: `Magento\Ui\Component\Form`.

Its most important function is to obtain data. As you can see, it checks whether the "id" parameter is set, and if so, adds a primary key filter.

```
public function getDataSourceData()
{
    $dataSource = [];

    $id = $this->getContext()->getRequestParam($this->getContext()->getDataProvider()->getRequestFieldName(), null);
    $filter = $this->filterBuilder->setField($this->getContext()->getDataProvider()->getPrimaryFieldName())
        ->setValue($id)
        ->create();
    $this->getContext()->getDataProvider()
        ->addFilter($filter);

    $data = $this->getContext()->getDataProvider()->getData();

    if (isset($data[$id])) {
```

```
$dataSource = [
    'data' => $data[$id]
];
} elseif (isset($data['items'])) {
    foreach ($data['items'] as $item) {
        if ($item[$item['id_field_name']] == $id) {
            $dataSource = ['data' => ['general' => $item]];
        }
    }
}
return $dataSource;
}
```

5.11 Form UiComponent | Usage Example

Form UiComponent | Usage Example

```
<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            ...
        </item>
        <item name="label" xsi:type="string" translate="true">Customer Information</item>
        <item name="layout" xsi:type="array">
            <item name="type" xsi:type="string">tabs</item>
            <item name="navContainerName" xsi:type="string">left</item>
        </item>
        <item name="buttons" xsi:type="array">
            ...
        </item>
    </argument>
```

[HOME](#)

Magento U

Notes:

This is an example from the Customer module of how the form UiComponent can be used.

Some things to note:

- **js_config:** Standard field for a UiComponent
- **label:** Self-explanatory
- **layout** Standard UiComponent directive
- **buttons:** This item includes button setup for a form. (The next slide provides more detail about this.)

5.12 Form UiComponent | Magento Block - Buttons

Form UiComponent | Magento Block - Buttons

```

<item name="back" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\BackButton </item>
<item name="delete" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\DeleteButton</item>
<item name="invalidateToken" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\InvalidateTokenButton</item>
<item name="resetPassword" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\ResetPasswordButton</item>
<item name="order" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\OrderButton</item>
<item name="reset" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\ResetButton</item>
<item name="save" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\SaveButton</item>
<item name="save_and_continue" xsi:type="string">
    Magento\Customer\Block\Adminhtml>Edit\SaveAndContinueButton</item>

```

Magento Block

HOME **Magento U**

Notes:

This example demonstrates how to configure a list of buttons. Each button is defined by an item element; its corresponding value is a Magento block that represents the required button.

The second code example, for the BackButton, represents a typical button example. As you analyze the code, you will see that getButtonData() returns an array with button configuration data like label, onClick handler, css class and sort_order.

getBackUrl is an example of button-specific functionality. For example, DeleteButton includes getDeleteUrl().

```

class BackButton extends GenericButton implements ButtonProviderInterface
{
    /**
     * @return array
     */
    public function getButtonData()
    {
        return [
            'label' => __('Back'),
            'on_click' => sprintf("location.href = '%s';", $this->getBackUrl()),
            'class' => 'back',
            'sort_order' => 10
        ];
    }
}

```

```
}

/**
 * Get URL for back (reset) button
 *
 * @return string
 */
public function getBackUrl()
{
    return $this->getUrl('*/*/');
}

}
```

5.13 Form UiComponent | Magento Block - Buttons

Form UiComponent | Magento Block - Buttons

```
graph TD; ButtonProvider[ButtonProvider] -- "implements" --> Button[Button]; ButtonProvider --> getButtonData[getButtonData()];
```

```
$customerId = $this->getCustomerId();
$canModify = !$customerId || !$this->customerAccountManagement->isReadOnly($this->getCustomerId());
$data = [];
if ($canModify) {
    $data = [
        'label' => ___('Save Customer'),
        'class' => 'save primary',
        'data_attribute' => [
            'mage-init' => ['button' => ['event' => 'save']],
            'form-role' => 'save',
        ],
        'sort_order' => 90,
    ];
}
return $data;
```

HOME

Magento U

Notes:

Continuing the code example, Button implements ButtonProvider, which is `Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface`.

Note the additional button properties included in the `data_attribute` element of the `getButtonData` result.

5.14 Form UiComponent | DataProvider

Form UiComponent | DataProvider

```
<dataSource name="customer_form_data_source">
    <argument name="dataProvider" xsi:type="configurableObject">
        <argument name="class" xsi:type="string">
            Magento\Customer\Model\Customer\DataProvider
        </argument>
        <argument name="name" xsi:type="string">customer_form_data_source</argument>
        <argument name="primaryFieldName" xsi:type="string">entity_id</argument>
        <argument name="requestFieldName" xsi:type="string">id</argument>
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="submit_url" xsi:type="url" path="customer/index/save"/>
                <item name="validate_url" xsi:type="url"
                    path="customer/index/validate"/>
            </item>
        </argument>
    </argument>
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="component"
xsi:type="string">Magento_Ui/js/form/provider</item>
        </item>
    </argument>
</dataSource>
```

Form
DataProvider

HOME

Magento U

Notes:

Here is a code example for a DataProvider definition for a customer form.

5.15 Form UiComponent | DataProvider Customer Example

Form UiComponent | DataProvider Customer Example

- Extends `\Magento\Ui\DataProvider\AbstractDataProvider`
- Implements `getData()` method, which generates data for the customer and customer's addresses
- Uses `eav` attributes meta data

[HOME](#)**Magento U****Notes:**

Some key points about Magento 2 form's `dataProvider`.

5.16 Adminhtml Forms | Fieldsets

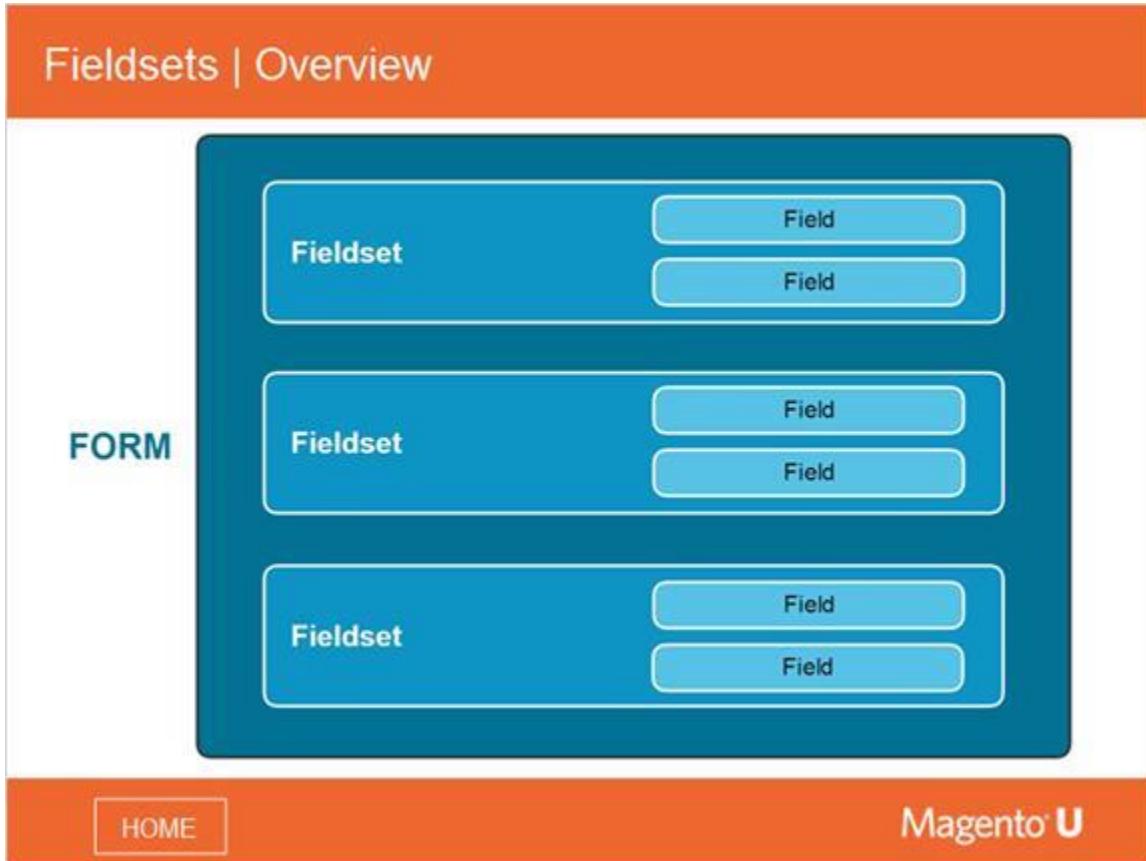
Adminhtml Forms | Fieldsets

Fieldsets

HOME

Magento U

5.17 Fieldsets | Overview



Notes:

As a reminder, this diagram represents a typical form structure. Each form consists of one or more fieldsets, and each fieldset consists of a group of fields.

5.18 Fieldsets | Code Definition

Fieldsets | Code Definition

```
<fieldset class="Magento\Ui\Component\Form\Fieldset">
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="component" xsi:type="string">
                Magento_Ui/js/form/components/fieldset</item>
        </item>
    </argument>
</fieldset>
```

[HOME](#)

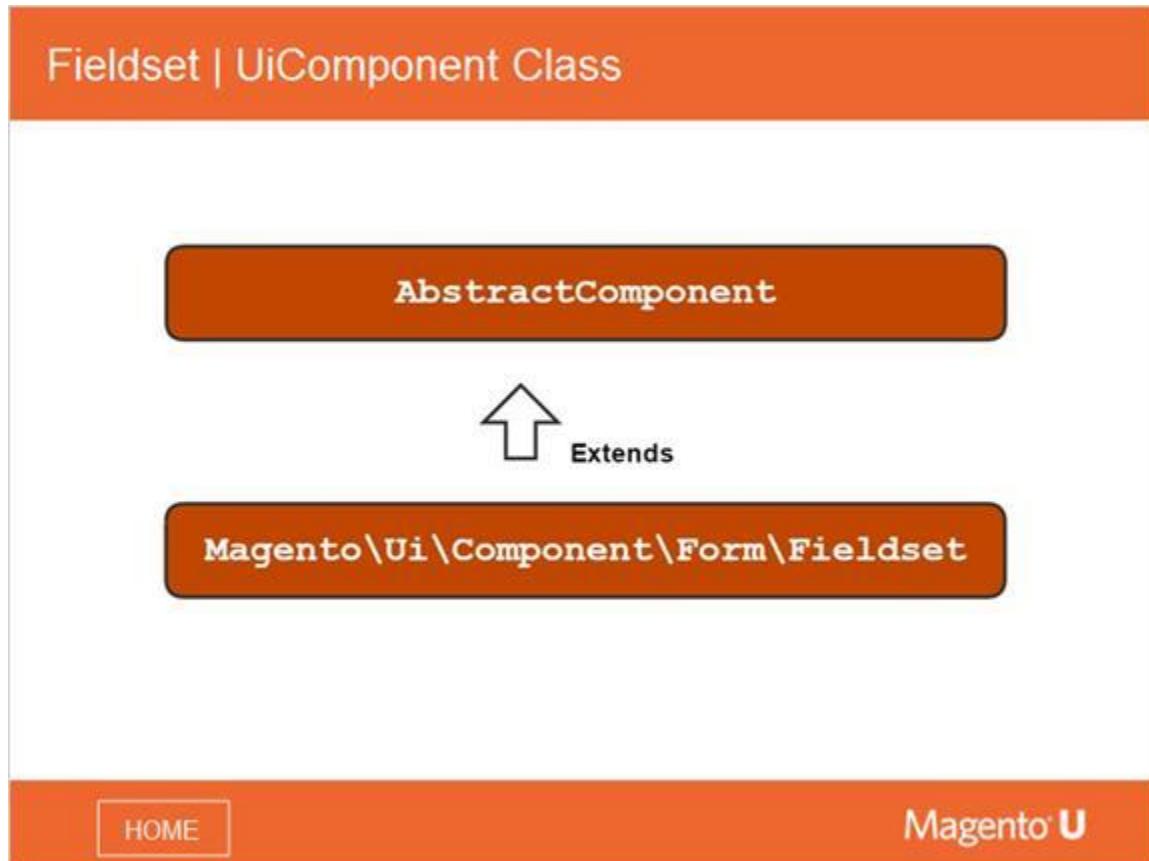
Magento U

Notes:

Here is the original definition of a fieldset as a Ui component.

As you can see, it is a pretty standard Ui Component that is implemented in a `Magento/Ui/view/base/web/js/form/components/fieldset.js` JavaScript module.

5.19 Fieldsets | UiComponent Class



Notes:

The fieldset component's PHP class extends `AbstractComponent`. This class is almost empty. Fieldset's configuration should be done in the form XML file, as we will see in the next slide.

5.20 Fieldsets | Customer Form Configuration Example

Fieldsets | Customer Form Configuration Example

```
<fieldset name="customer">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="label" xsi:type="string" translate="true">
                Account Information</item>
            </item>
        </argument>
        <field name="entity_id">
            ...
        </field>
        <field name="created_in">
            ...
        </field>
        ...
    </argument>
</fieldset>
```

[HOME](#)

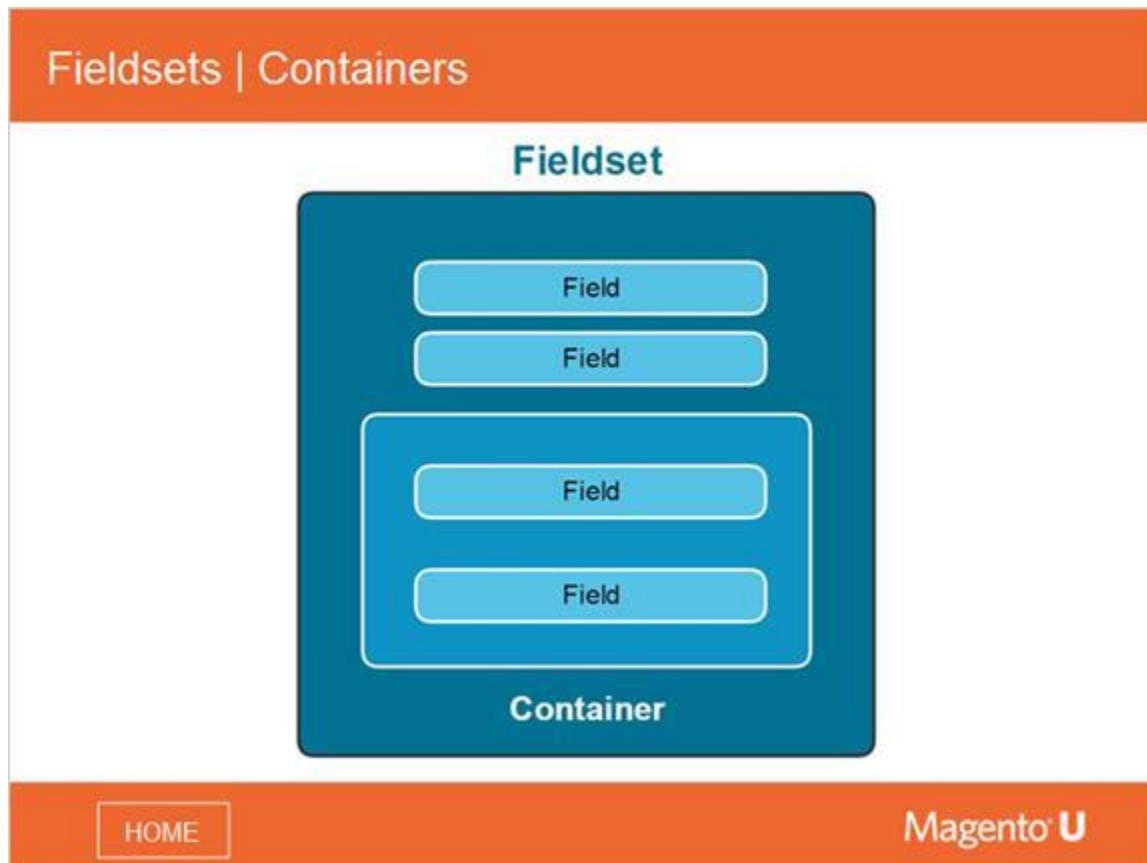
Magento U

Notes:

The slide provides a customer form configuration example.

This form has many fieldsets, with eachfieldset encapsulating field elements. As we see here,fieldset configuration consists of a data argument and a set of field nodes. Each node includes field configuration, which will be described in the slides that follow.

5.21 Fieldsets | Containers



Notes:

The role of a container is to group multiple fields inside of a fieldset. The next slide provides an example of when containers could be useful.

5.22 Fieldsets | Interface

The screenshot shows the 'Account Information' section of the Magento 2 Admin Panel. At the top right, there is a red callout box labeled 'Group Container' pointing to a group of fields. Inside this group, there is a dropdown labeled 'Group' with 'General' selected, and a checkbox labeled 'Disable Automatic Group Change Based on VAT ID'. Below this group, there are fields for 'Prefix' and 'First Name' (with 'Veronica' entered). The 'Associate to Website' dropdown is set to 'Main Website'. The 'Group Container' label is overlaid on the top right corner of the group of fields.

Account Information

Associate to Website • Main Website ?

Group • General ▾

Disable Automatic Group Change Based on VAT ID

Prefix

First Name • Veronica

Group Container

HOME

Magento U

Notes:

You can see two fields grouped here: the dropdown "General" and the checkbox "Disable Automatic Group Change Based on VAT ID." Both are contained within the same label, "Group."

5.23 Fieldsets | Container Definition

Fieldsets | Container Definition

```
<container class="Magento\Ui\Component\Container">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">uiComponent</item>
        </item>
        <item name="template" xsi:type="string">templates/container/default</item>
    </argument>
</container>
```

[HOME](#)**Magento U****Notes:**

This code demonstrates a declaration of a Container UiComponent definition in `definition.xml`.

5.24 Fieldsets | Container Configuration Example

Fieldsets | Container Configuration Example

```
<container name="container_group">
    <argument name="data" xsi:type="array">
        <item name="type" xsi:type="string">group</item>
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">
                Magento_Ui/js/form/components/group</item>
            <item name="label" xsi:type="string" translate="true">Group</item>
            <item name="required" xsi:type="boolean">true</item>
            <item name="dataScope" xsi:type="boolean">false</item>
            <item name="sortOrder" xsi:type="number">20</item>
            <item name="validateWholeGroup" xsi:type="boolean">true</item>
        </item>
    </argument>
    <field name="group_id">
        ...
    </field>
    <field name="disable_auto_group_change">
        ...
    </field>
</container>
```

[HOME](#)

Magento U

Notes:

This piece of XML code defines a group, the same as shown on the fieldsets interface diagram two slides earlier.

You can see it includes two fields and a set of parameters in the data argument.

5.25 Fieldsets | Container Class

Fieldsets | Container Class

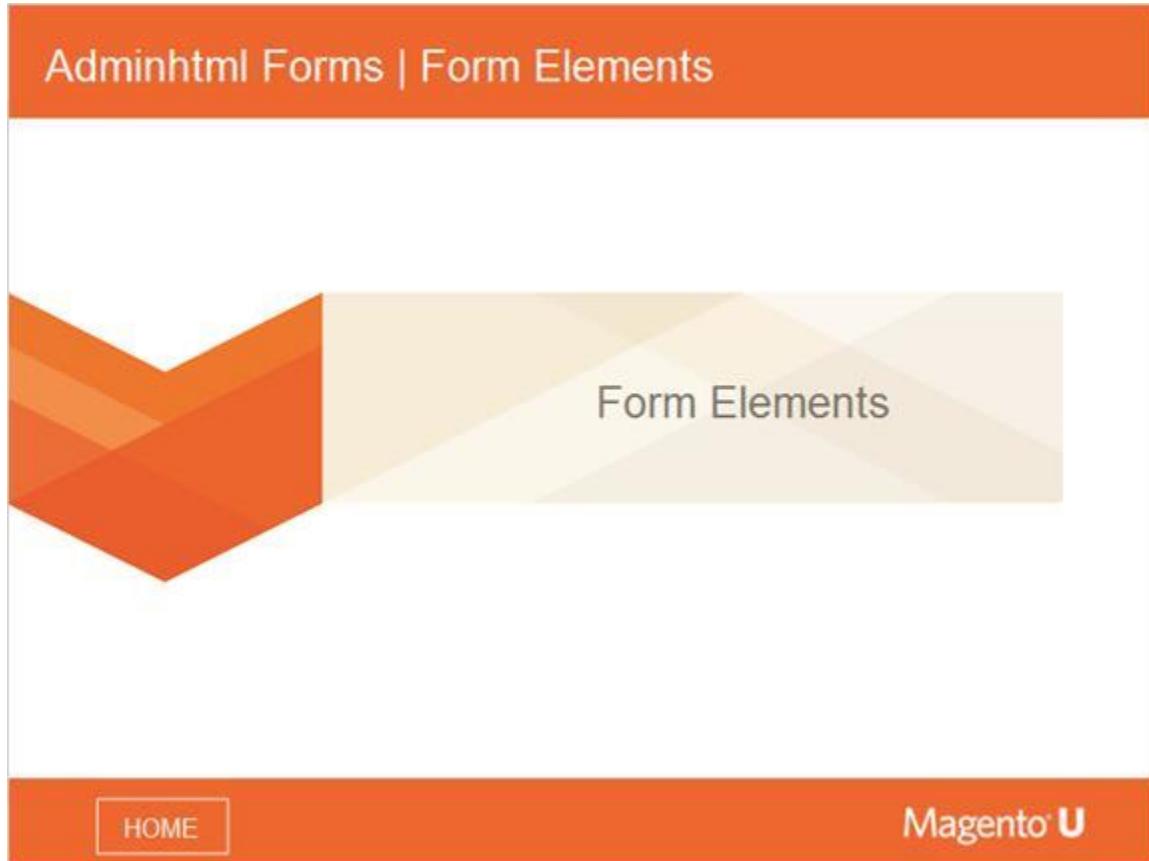
```
class Container extends AbstractComponent
{
    const NAME = 'container';

    /**
     * Get component name
     *
     * @return string
     */
    public function getComponentName()
    {
        $type = $this->getData('type');
        return static::NAME . ($type ? ('.' . $type): '');
    }
}
```

[HOME](#)**Magento U****Notes:**

Here is an example of a Container PHP class. As you can see, it is relatively trivial.

5.26 Adminhtml Forms | Form Elements

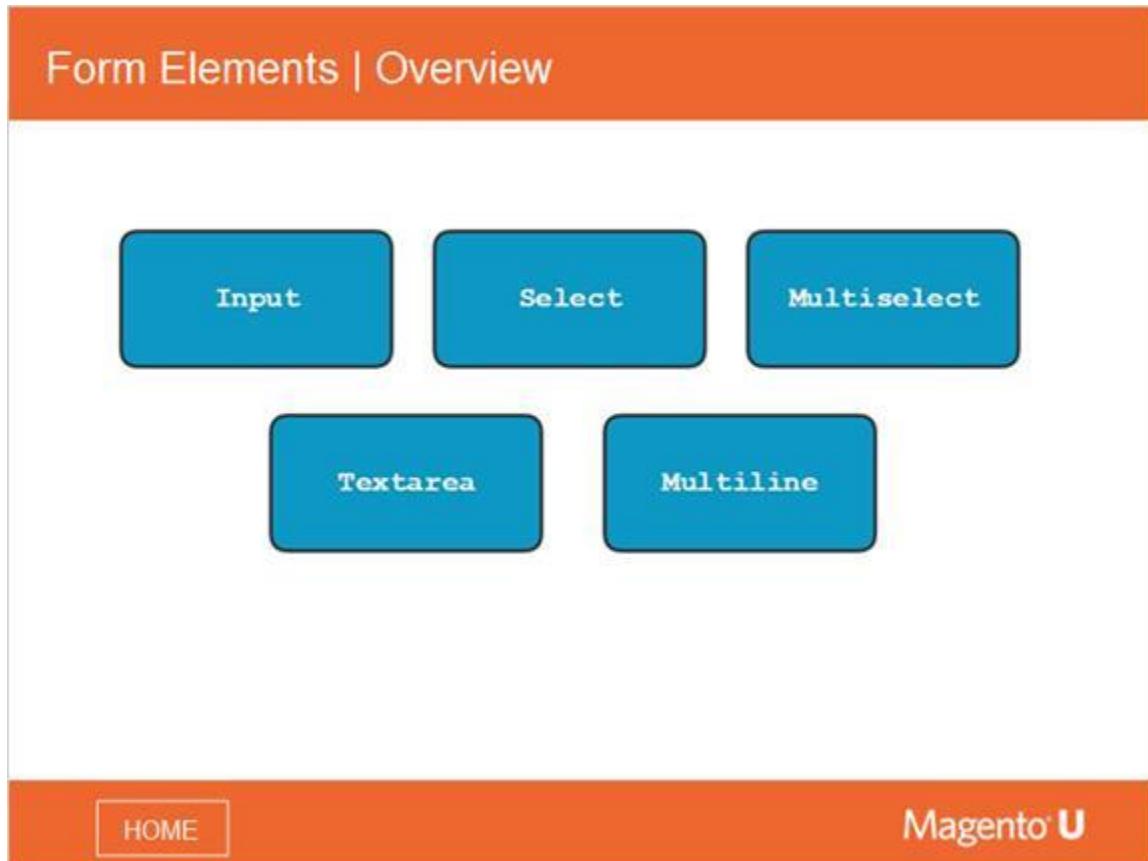


The slide has an orange header bar with the text "Adminhtml Forms | Form Elements". Below the header is a large graphic featuring a stylized orange chevron shape on the left and a light beige background with a subtle diagonal striped pattern on the right. The text "Form Elements" is centered in the striped area. At the bottom left is a white rectangular button with the word "HOME" in black. At the bottom right is the "Magento U" logo.

Notes:

And now, the final topic of this module: form elements.

5.27 Form Elements | Overview



Notes:

This diagram presents the most important form elements. You can find the full list in the `definition.xml` file.

5.28 Form Elements | Data Types

The slide displays a grid of eight data types arranged in two columns and four rows. The first column contains 'Text' (top), 'Number' (second), 'Price' (third), and 'Image' (bottom). The second column contains 'File' (top), 'Date' (second), 'Boolean' (third), and 'Email' (bottom). All boxes are blue with white text.

Form Elements | Data Types

Text	File
Number	Date
Price	Boolean
Image	Email

HOME Magento U

Notes:

The slide shows a list of data types.

Note that data type does not explicitly define control, but rather how to treat data related to a certain field.

As an example, here is the code for an abstract data type class:

```
abstract class AbstractDataType extends AbstractComponent implements DataTypeInterface
{
    /**
     * Validate value
     *
     * @return bool
     */
    public function validate()
    {
        return true;
    }
}
```

5.29 Form Elements | Definition Example

Form Elements | Definition Example

```
<input class="Magento\Ui\Component\Form\Element\Input">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="component" xsi:type="string">
                Magento_Ui/js/form/element/abstract</item>
            <item name="template" xsi:type="string">ui/form/field</item>
        </item>
    </argument>
</input>
```

[HOME](#)**Magento U**

Notes:

Here we can see how form elements are defined in the definition.xml file.

There are three important elements to note:

- **Component PHP class:** Magento\Ui\Component\Form\Element\Input
- **JavaScript module:** MagentoUi/view/base/web/js/form/element/abstract.js
- **Template:** MagentoUi/view/base/web/templates/form/element/input.html

5.30 Form Elements | Generic Field Template

Form Elements | Generic Field Template

- Generic field template is:
[Magento/Ui/view/base/web/templates/form/field.html](#)
- Includes generic form element layout.

[HOME](#)

Magento U

Notes:

A generic field template plays the same role as containers did in Magento 1. It wraps an element's template with auxiliary components like before/after triggers.

You can take a look at the corresponding code below:

```
<div class="admin_field"
    visible="visible"
    css="$data.additionalClasses"
    attr="'data-index': index">
    <label class="admin_field-label" if="$data.label" visible="$data.labelVisible" attr="for: uid">
        <span text="label" attr="data-config-scope": $data.scopeLabel"/>
    </label>
    <div class="admin_field-control"
        css="'_with-tooltip': $data.tooltip, '_with-reset': $data.showFallbackReset &&
$data.isDifferedFromDefault">
        <render args="elementTmp1" ifnot="hasAddons()" />
        <div class="admin_control-addon" if="hasAddons()">
            <render args="elementTmp1"/>
        <label class="admin_addon-prefix" if="$data.addbefore" attr="for: uid">
```

```
        <span text="addbefore"/>
    </label>
    <label class="admin__addon-suffix" if="$data.addafter" attr="for: uid">
        <span text="addafter"/>
    </label>
</div>

<render args="tooltipTpl" if="$data.tooltip"/>

<render args="fallbackResetTpl" if="$data.showFallbackReset && $data.isDifferedFromDefault"/>

<label class="admin__field-error" if="error" attr="for: uid" text="error"/>

<div class="admin__field-note" if="$data.notice" attr="id: noticeId">
    <span text="notice"/>
</div>

<div class="admin__additional-info" if="$data.additionalInfo" html="$data.additionalInfo"></div>

    <render args="$data.service.template" if="$data.hasService()"/>
</div>
</div>
```

5.31 Form Elements | Element Template

Form Elements | Element Template

```
<input class="admin__control-text" type="text"
    data-bind=""
        event: {change: userChanges},
        value: value,
        hasFocus: focused,
        valueUpdate: valueUpdate,
        attr: {
            name: inputName,
            placeholder: placeholder,
            'aria-describedby': noticeId,
            id: uid,
            disabled: disabled
        }"/>
```

[HOME](#)

Magento U

Notes:

This code presents an example of an element's template using `input.html`, which was included in the `definition.xml` slide shown earlier.

As you can see, it is a knockout template composed of certain data (`value`, `name`, `id`) that are inserted using the generic `UiComponent` engine.

5.32 Form Elements | Configuration Example

Form Elements | Configuration Example

```
<field name="prefix">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="visible" xsi:type="boolean">true</item>
            <item name="dataType" xsi:type="string">text</item> Data Type
            <item name="formElement" xsi:type="string">input</item> Form Element
            <item name="source" xsi:type="string">customer</item>
        </item>
    </argument>
</field>
```

[HOME](#)**Magento U**

Notes:

Finally, we can take a look at the practical application of the theory above.

This slide shows an example of real input form field declaration for the customer module.

Note the two major configuration options that define the template's look and feel:

- **dataType**
- **formElement**

5.33 Form Elements | Select Example

Form Elements | Select Example

```
<field name="website_id">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="dataType" xsi:type="string">number</item>
            <item name="formElement" xsi:type="string">select</item>
            <item name="source" xsi:type="string">customer</item>
            <item name="validation" xsi:type="array">
                <item name="required-entry" xsi:type="boolean">true</item>
            </item>
            <item name="tooltip" xsi:type="array">
                <item name="link" xsi:type="string">
http://www.magentocommerce.com/knowledge-base/entry/understanding-store-scopes
                </item>
                <item name="description" xsi:type="string" translate="true">
                    If your Magento site has multiple views, you can
                    set the scope to apply to a specific view.
                </item>
            </item>
        </item>
    </argument>
</field>
```

HOME

Magento U

Notes:

Another example, this time for the select field.

You can see more configuration options in this example, such as validation (which defines the CSS class used for validation) and tooltip (which shows a message on mouse-over).

Note that dataType for select-type fields defines the type of select values.

5.34 Reinforcement Exercise (6.5.1)

Reinforcement Exercise (6.5.1): Add a Form

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

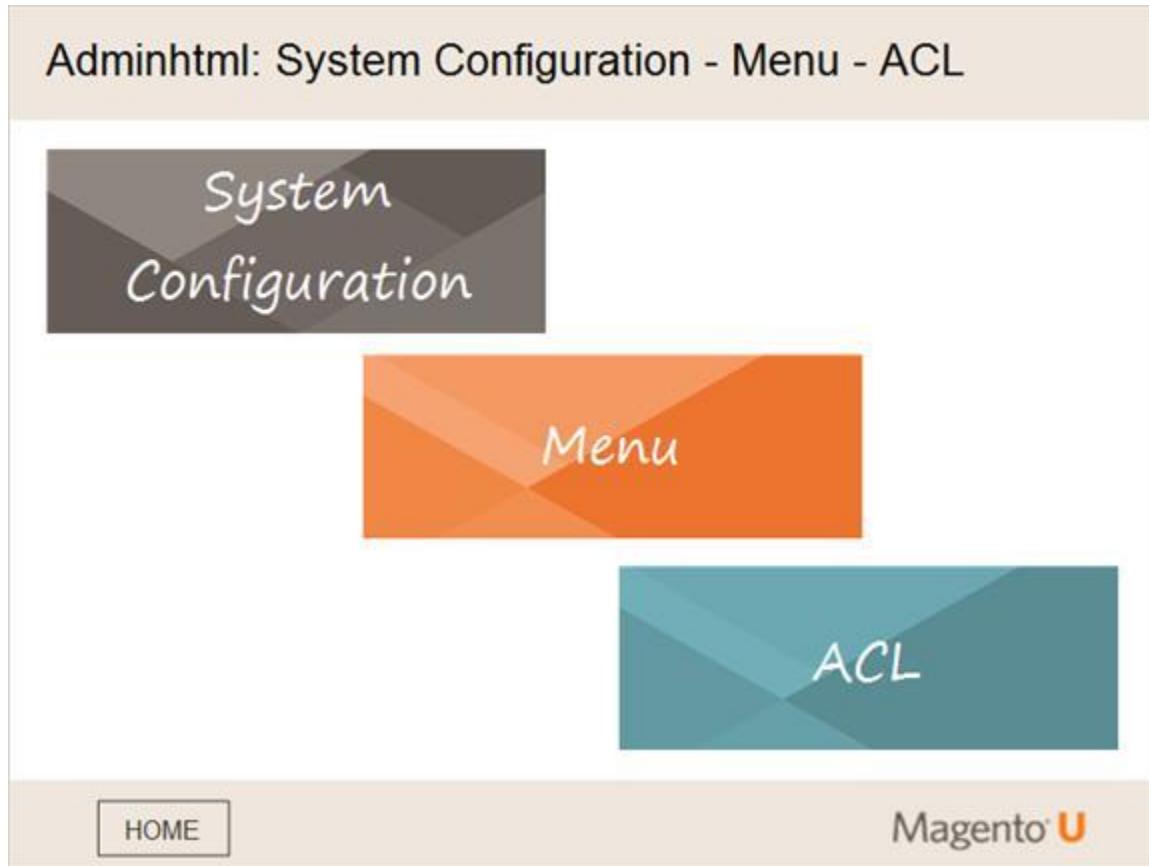
Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

6. Configuration

6.1 Adminhtml: System Configuration - Menu - ACL

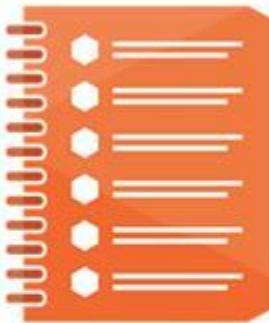


Notes:

Now that we have finished our general discussion of the framework API, we are going to look at the API in more detail, focused on repositories and the business logic API.

6.2 Module Topics | System Configuration - Menu - ACL

Module Topics | System Configuration - Menu - ACL



In this module, we will discuss in detail...

- System configuration
- Menus
- ACL permissions

[HOME](#)

Magento U

Notes:

In this module, we will discuss in detail...

- System configuration
- Menus
- ACL permissions

6.3 System Configuration | Definition

The screenshot shows a section titled "System Configuration | Definition". A large orange callout bubble points from the right towards the text "System configuration...". Below the bubble, there is descriptive text about system configuration in Magento 2. At the bottom of the screenshot, there is a navigation bar with "HOME" and the "Magento U" logo.

System Configuration | Definition

System
configuration...

System configuration in Magento 2 allows you to quickly and easily set up form elements to configure your system and custom modules.

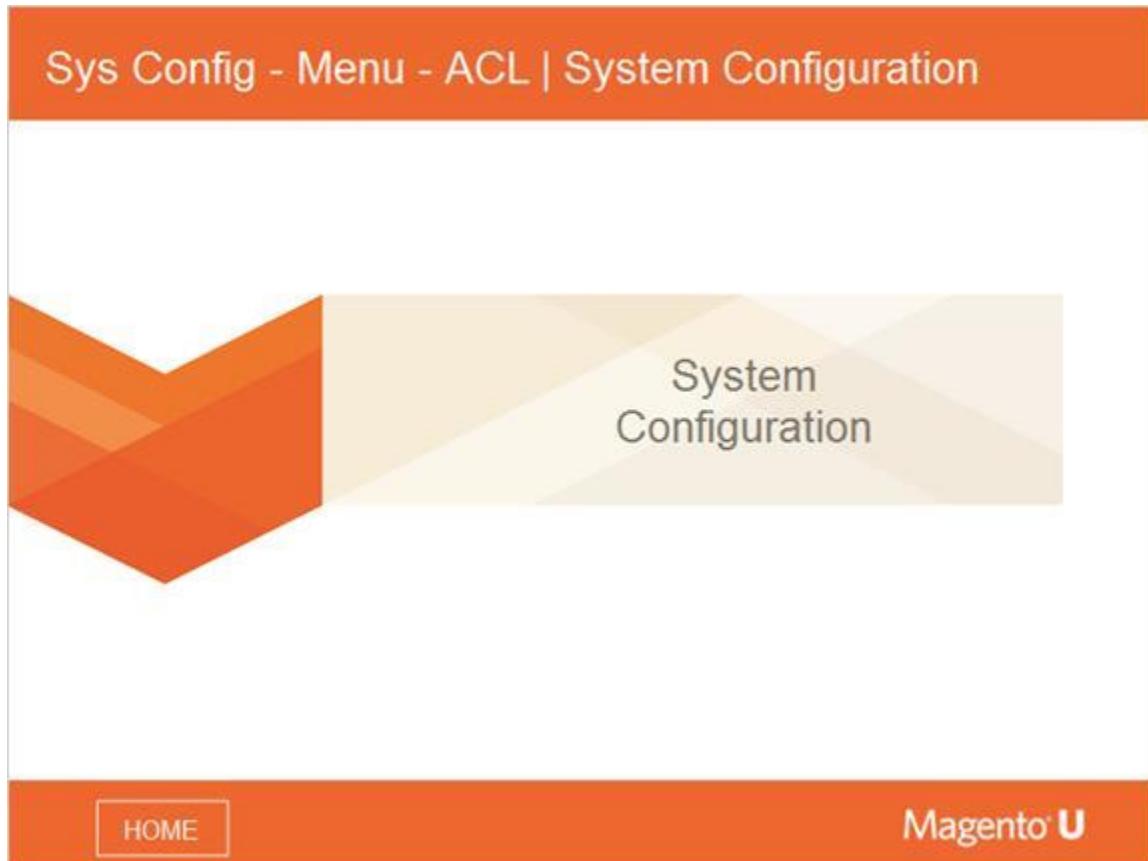
HOME

Magento U

Notes:

System configuration in Magento 2 allows you to quickly and easily set up form elements to configure your system and custom modules.

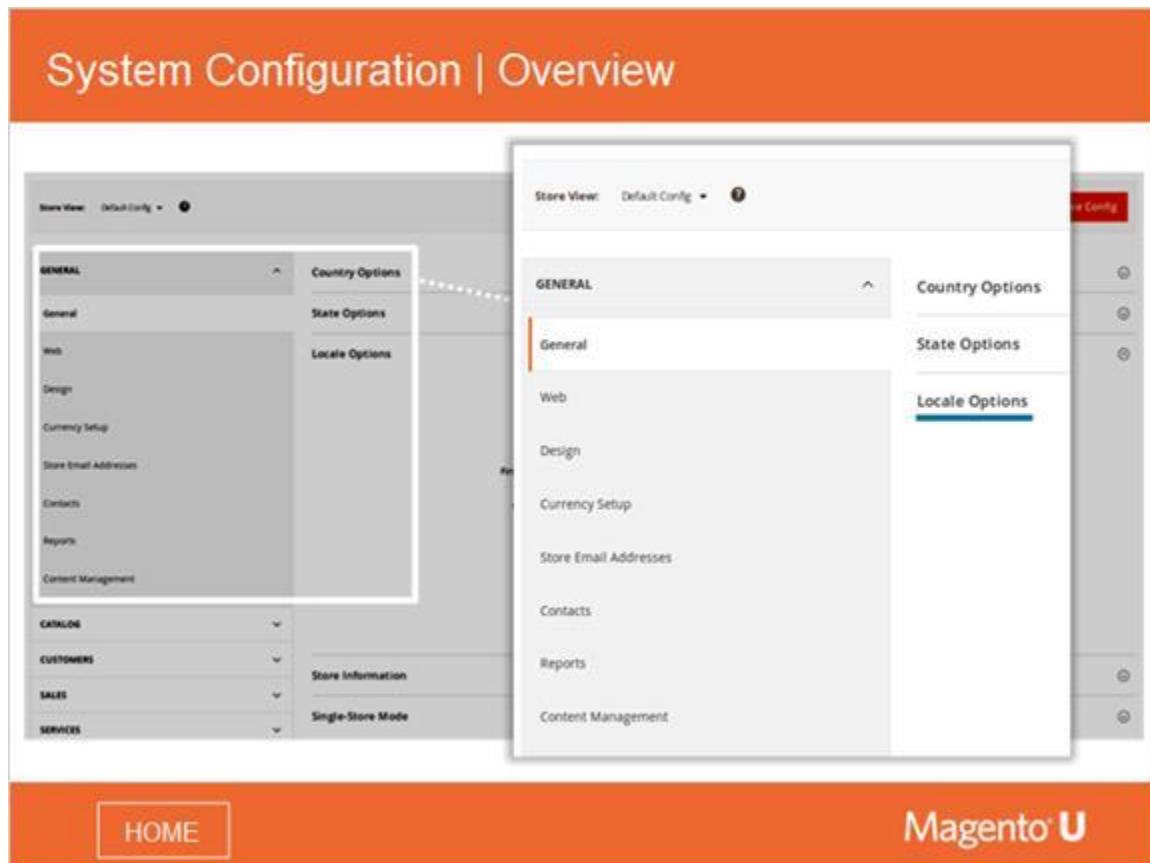
6.4 Sys Config - Menu - ACL | System Configuration



Notes:

We'll begin with a discussion of system configuration within Magento 2.0.

6.5 System Configuration | Overview



Notes:

The system configuration page in Magento 2 provides a list of available options.

Here are some important things to note:

- Configuration options are for the merchant, so technical aspects that the merchant may not understand are not included. For example, the order of **calculating** totals for a shopping cart is technical information and is not presented in system configuration, while the order of **displaying** totals is included.
- Every configuration option value from this page is stored in a database (unlike XML configuration). The corresponding table is `core_config_data`, as in Magento 1.
- Configuration options can be scoped, on global, website, and store levels; scope changing is possible by selecting a scope in the top left dropdown of the page.
- All configuration options are previously declared in a special configuration file: `system.xml`.
- Option names consist of 3 parts: section, group, field ID.
For example, the "Locale" option presented on this slide will have the name: `general/locale/code`.

6.6 System Configuration | system.xml

System Configuration | system.xml

```

<config xmlns:xsi="...">
    <system>
        <tab id="catalog" translate="label" sortOrder="200">
            <label>Catalog</label>
        </tab>
        <section id="catalog" ...>
            <class>separator-top</class>
            <label>Catalog</label>
            <tab>catalog</tab>          Parent Tab
            <resource>Magento_Catalog::config_catalog</resource>
            <group id="fields_masks" ...>
                <label>Product Fields Auto-Generation</label>
                <field id="sku" translate="label comment" type="text" ...>
                    <label>Mask for SKU</label>
                    <comment>Use {{name}} as Product Name placeholder</comment>
                </field>
                ...
            </group>                      Field Configuration
        </section>
    </system>
</config>

```

Tab Definition

CSS Class

ACL Resource

Parent Tab

Field Configuration

HOME

Magento U

Notes:

On this page, we can see a typical configuration for the system configuration. Let's explore its structure.

First of all, configuration is located in etc/adminhtml/system.xml. Its XSD schema is located in Magento/Config/etc/system_file.xsd

Two types of second-level nodes are available in the system.xml file: tab and section.

In the tab node, you can define a new tab for the configuration. The example on the slide demonstrates the "Catalog" tab definition.

The section node is where most of the options are described. It defines (1) which tab the section belongs to (in this case, catalog), and (2) the resource (ACL resource and group). The section node also has a name (second-level after Tab).

Group provides third-level option groupings (after Tab). It is only displayed on the right side of a page, not the left side (called an "accordion"). Each group belongs to the section it is declared in (as does the tab), has its own label, and has a list of fields.

Field is actually an option in itself. It might have different scope (defined by the field node attributes, discussed later). Each field has its name (id attribute), label (<label> child node), and type (type attribute).

You can also optionally include other child subnodes like source_model and frontend_model. They will be covered later in the course as well.

6.7 System Configuration | Field Attributes

System Configuration | Field Attributes

```
<field  
    id="..."  
    translate="label"  
    type="select"  
    sortOrder="1"  
    showInDefault="1"  
    showInWebsite="1"  
    showInStore="1">
```

[HOME](#)

Magento U

Notes:

Let's go deeper into the structure of `system.xml`. On the slide, you can see the following field node attributes:

- **Id** - A field name. It is a third part of an option's name (after section and group).
- **Translate** - A standard translation attribute. As in Magento 1, it defines which child node to translate.
- **Type** - Defines what kind of control has to be used to render an option. On the next slide you will see the available types.
- **SortOrder** - The order inside an accordion.
- **showIn*** - Defines a scope.

Example:

If `showInDefault=1, showInWebsite=0, showInStore=0`, the option will be global and will not be visible on the website/store level.

If `showInDefault=1, showInWebsite=1, showInStore=0`, the option will be visible on default and website scopes, but its scope will be website (which means you can define a default value but then you can redefine the value on the website level).

6.8 System Configuration | Field Types

System Configuration | Field Types

Text Select

Image Textarea

HOME

Magento U

Notes:

Here is a list of standard field types. For non-standard fields, you can use the `frontend_model` child node, which is described later.

6.9 System Configuration | Field Children

System Configuration | Field Children

```
<xs:element name="label" type="xs:string" />
<xs:element name="comment" type="xs:string" />
<xs:element name="tooltip" type="xs:string" />
<xs:element name="hint" type="xs:string" />
<xs:element name="frontend_class" type="xs:string" />
<xs:element name="frontend_model" type="typeModel" />
<xs:element name="backend_model" type="typeModel" />
<xs:element name="source_model" type="typeModel" />
<xs:element name="config_path" type="typeConfigPath" />
<xs:element name="validate" type="xs:string" />
<xs:element name="can_be_empty" type="xs:int" />
<xs:element name="if_module_enabled" type="typeModule" />
<xs:element name="base_url" type="typeUrl" />
<xs:element name="button_url" type="typeUrl" />
<xs:element name="more_url" type="typeUrl" />
<xs:element name="demo_url" type="typeUrl" />
<xs:element name="button_label" type="xs:string" />
<xs:element name="hide_in_single_store_mode" type="xs:int" />
<xs:element name="upload_dir" type="typeUploadDir" />
<xs:element ref="source_service" />
<xs:element ref="options" />
<xs:element ref="depends" />
<xs:element ref="attribute" />
<xs:element ref="requires" />
```

[HOME](#)**Magento U**

Notes:

The slide shows a list of all the available `<field>` node children. The list is taken from an xsd file: `Magento/Config/etc/system_file.xsd`.

Note the following:

- **label:** The visible name of an option.
- **comment:** Text written in a small font below an option input. For example: Catalog/Storefront/Products per Page on Grid Allowed Values; a comment is “comma-separated”.
- **tooltip:** Draws a small question mark, which shows a message on mouse hover. For example, see General/Design/Design Theme/User-agent exceptions; it is defined in `Magento/Backend/etc/adminhtml/system.xml`.
- **hint:** Not used in the native Magento.
- **frontend_class:** CSS class to assign to the option. For example: General/Store Information/Country, defined in `Magento/Backend/etc/adminhtml/system.xml`.
- **frontend_model:** Used to define custom content for an option; will be discussed in more detail later in the course.
- **backend_model:** A class which allows an action to be performed when an option is saved. For example: class `Magento\Config\Model\Config\Backend\Email\Address`.
- **source_model:** Used to provide a list of values for select-type configuration values.
- **config_path:** Defines an alternative path to the option, instead of the standard three-levels path. This alternate path is used to save an option in a database; for example: `Magento/Paypal/etc/adminhtml/system.xml`, `merchant_country` field.
- **validate:** CSS class used for validation. For example: General/Store Email Addresses/Custom Email 1/Sender Email, defined in `Magento/Backend module`.

- **can_be_empty**: Flag which defines whether an option can have no value.
- **if_module_enabled**: Inherited from Magento 1, a condition that defines an option only when particular modules are enabled.
- **base_url**: Used for image-type fields.
- **button_url, more_url**: Specific properties; in the native Magento, used in the PayPal module.
- **demo_url**: Not used in the native Magento.
- **button_label**: A specific property; used only with a custom frontend model (for example, in the PayPal module).
- **hide_in_single_store_mode**: A property that defines whether a field should be available for a single store mode. For example, price_scope doesn't make sense in such a mode.
- **upload_dir**: Used for image-type fields.
- **source_service**: Not currently used in Magento.
- **depends**: Defines other nodes on which this node depends. For example, see `Magento/Paypal/etc/adminhtml/system/express_checkout.xml`.
- **attribute**: Allows the definition of a custom field's attribute. For example, see `Magento/Paypal/etc/adminhtml/system/express_checkout.xml`, and `Magento/Paypal/Block/Adminhtml/System/Config/ApiWizard.php`.
- **requires**: Makes a field available only if another field/group is activated. For example, look at the PayPal module `Magento/Paypal/etc/adminhtml/system/express_checkout.xml`.

6.10 System Configuration | Source Model Configuration

System Configuration | Source Model Configuration

```
<field id="list_mode"
    translate="label"
    type="select"
    sortOrder="1"
    showInDefault="1"
    showInWebsite="1"
    showInStore="1">

    <label>List Mode</label>
    <source_model>
        Magento\Catalog\Model\Config\Source\ListMode
    </source_model>
</field>
```

[HOME](#)

Magento U

Notes:

Here is a code example of a source model configuration.

As mentioned earlier, the source model is used to provide a list of available values for dropdown-type configuration options. On the next slide is an example of such a model implementation.

6.11 System Configuration | Source Model Example

System Configuration | Source Model Example

```
namespace Magento\Catalog\Model\Config\Source;

class ListMode implements \Magento\Framework\Option\ArrayInterface
{
    /**
     * {@inheritDoc}
     */
    * @codeCoverageIgnore
    */
    public function toOptionArray()
    {
        return [
            ['value' => 'grid', 'label' => __('Grid Only')],
            ['value' => 'list', 'label' => __('List Only')],
            ['value' => 'grid-list', 'label' => __('Grid (default) / List')],
            ['value' => 'list-grid', 'label' => __('List (default) / Grid')]
        ];
    }
}
```

[HOME](#)**Magento U**

Notes:

Here we can see an example of a source model.

The main goal of these classes is to provide a list of available options for select-type configuration fields.

A class has to implement `\Magento\Framework\Option\ArrayInterface`, which requires the implementation of the `toOptionArray()` method.

This method should return an array of values and labels in the format shown. Note that the “`__`” (double underscore) is now a global function that provides translation.

6.12 System Configuration | Frontend Model Configuration

System Configuration | Frontend Model Configuration

```
<field
    id="date_fields_order"
    translate="label"
    type="select"
    sortOrder="2"
    showInDefault="1"
    showInWebsite="1"
    showInStore="1">

    <label>Date Fields Order</label>
    <frontend_model>
        Magento\Catalog\Block\Adminhtml\Form\Renderer\Config\DateFieldsOrder
    </frontend_model>
</field>
```

[HOME](#)

Magento U

Notes:

The code provides an example of a frontend model configuration. A frontend model is used to render a custom element for a field.

6.13 System Configuration | Frontend Model Implementation

System Configuration | Frontend Model Implementation

```
class DateFieldsOrder extends Field
{
    // @param AbstractElement $element
    // @return string
    // @SuppressWarnings(PHPMD.NPathComplexity)
    protected function _getElementHtml(AbstractElement $element)
    {
        $_options = ['d' => __('Day'), 'm' => __('Month'), 'y' => __('Year')];
        $element->setValues($_options)->setClass('select-date')->setName($element
            ->getName() . '[]');

        if ($element->getValue()) {
            $values = explode(',', $element->getValue());
        } else {
            $values = [];
        }

        $_parts = [];
        $_parts[] = $element->setValue(isset($values[0]) ? $values[0] : null)->getElementHtml();
        $_parts[] = $element->setValue(isset($values[1]) ? $values[1] : null)->getElementHtml();
        $_parts[] = $element->setValue(isset($values[2]) ? $values[2] : null)->getElementHtml();
        return implode(' <span></span> ', $_parts);
    }
}
```

[HOME](#)
[Magento U](#)

Notes:

Here is an example of a frontend model implementation. You can see that it extends `Magento\Config\Block\System\Config\Form\Field`, which itself extends `\Magento\Backend\Block\Template`. Note that the overridden `_getElementHtml()` method generates the “value” part of an option (control).

You can better understand how the frontend model works by looking at the `render()` method (defined in the `Magento\Config\Block\System\Config\Form\Field` class). The highlighted code shows which methods can be overridden in the frontend model, and what effect that would have.

```
public function render(\Magento\Framework\Data\Form\Element\AbstractElement $element)
{
    $isCheckboxRequired = $this->_isInheritCheckboxRequired($element);

    // Disable element if value is inherited from other scope. Flag has to be set before the value is
    rendered.
    if ($element->getInherit() == 1 && $isCheckboxRequired) {
        $element->setDisabled(true);
    }

    $html = '<td class="label"><label for="' .
        $element->getHtmlId() .
        '">' .
```

6. Configuration

```
        $element->getLabel() .  
        '</label></td>';  
    $html .= $this->_renderValue($element);  
  
    if ($isCheckboxRequired) {  
        $html .= $this->_renderInheritCheckbox($element);  
    }  
  
    $html .= $this->_renderScopeLabel($element);  
    $html .= $this->_renderHint($element);  
  
    return $this->_decorateRowHtml($element, $html);  
}  
  
protected function _renderValue(\Magento\Framework\Data\Form\Element\AbstractElement $element)  
{  
    if ($element->getTooltip()) {  
        $html = '<td class="value with-tooltip">';  
        $html .= $this->_getElementHtml($element);  
        $html .= '<div class="tooltip"><span class="help"><span></span></span></div>';  
        $html .= '<div class="tooltip-content">' . $element->getTooltip() . '</div></div>';  
    } else {  
        $html = '<td class="value">';  
        $html .= $this->_getElementHtml($element);  
    }  
    if ($element->getComment()) {  
        $html .= '<p class="note"><span>' . $element->getComment() . '</span></p>';  
    }  
    $html .= '</td>';  
    return $html;  
}  
...  
}
```

The `$element` object (an instance of the `AbstractElement` class) is an instance of a field configured in `system.xml`, so we have access to its properties. For example, in the next example (from the Magento/Paypal module), you can see the use of field properties from `system.xml`:

```
namespace Magento\Paypal\Block\Adminhtml\System\Config;  
  
class BmlApiWizard extends ApiWizard  
{  
    /**  
     * Path to block template  
     */  
    const WIZARD_TEMPLATE = 'system/config/bml_api_wizard.phtml';  
  
    /**  
     * Get the button and scripts contents  
     *  
     * @param \Magento\Framework\Data\Form\Element\AbstractElement $element  
     * @return string  
     */  
    protected function _getElementHtml(\Magento\Framework\Data\Form\Element\AbstractElement $element)  
    {  
        $originalData = $element->getOriginalData();  
        $this->addData(  
            [  
                'button_label' => __($originalData['button_label']),  
            ]  
        );  
    }  
}
```

```
        'button_url' => $originalData['button_url'],
        'html_id' => $element->getHtmlId(),
    ]
);
return $this->_toHtml();
}
}
```

6.14 Reinforcement Exercise (6.6.1)

Reinforcement Exercise (6.6.1): System Configuration

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

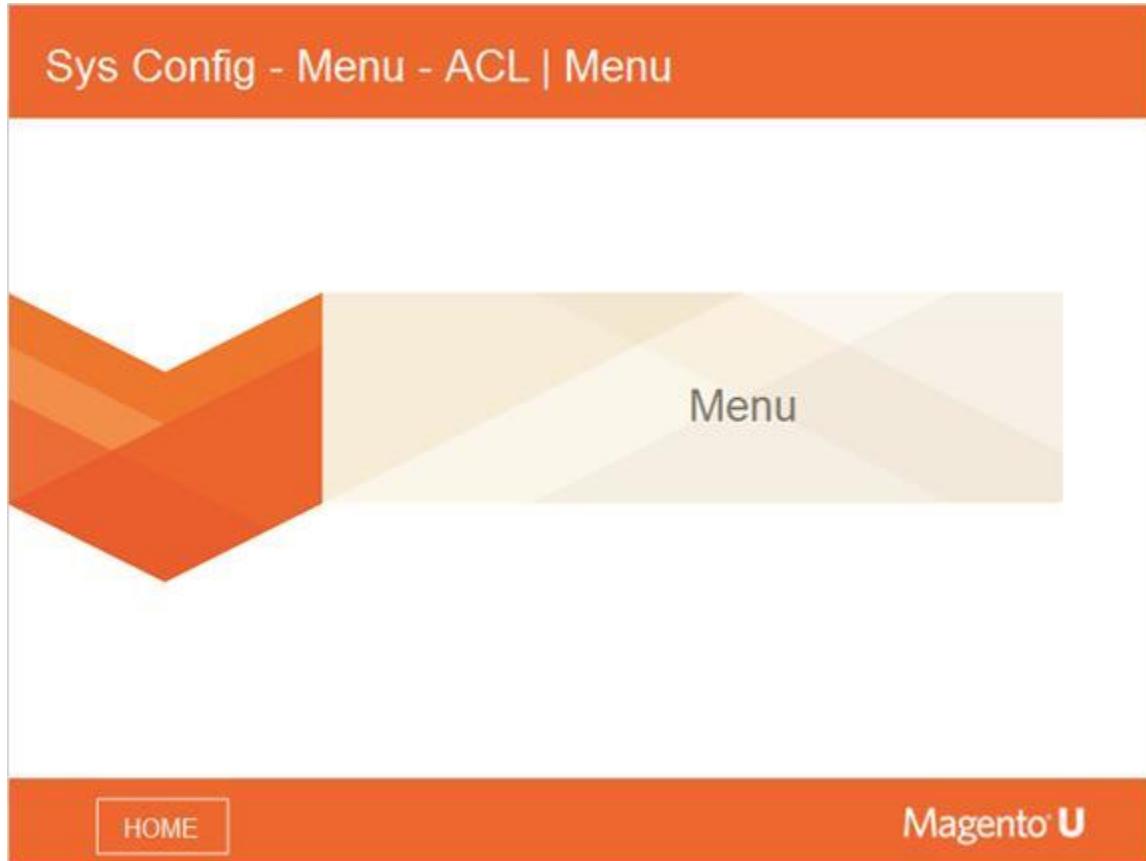
HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

6.15 Sys Config - Menu - ACL | Menu



Notes:

Next, we'll look at the Menu feature within Magento 2.0.

6.16 Menu | Overview

The screenshot shows the Magento 2 Admin Panel interface. At the top, there's a navigation bar with 'Menu | Overview'. Below it is a sidebar with various icons and labels: Dashboard, PRODUCTS (with 94 items), CUSTOMERS, MARKETING, CONTENT, REPORTS, STORES, and SYSTEM. The 'STORES' icon is highlighted with a red box. A modal window titled 'Stores' is open, listing several configuration options: Settings, All Stores, Configuration, Terms and Conditions, Order Status, Taxes, Tax Rules, Tax Zones and Rates, Currency, Currency Rates, and Currency Symbols. The 'Currency' option is also highlighted with a red box. At the bottom of the screen, there are 'HOME' and 'Magento U' buttons.

Notes:

Unlike Magento 1, the menu in Magento 2 is located on the left side of a page. This is done to simplify access to a menu from a tablet or smartphone.

Other than that, the menu structure in both Magento 1 and Magento 2 are the same.

6.17 Menu | menu.xml

Menu | menu.xml

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ...>
    <menu>
        <add id="Magento_Catalog::catalog"
            title="Products"
            module="Magento_Catalog"
            sortOrder="20"
            dependsOnModule="Magento_Catalog"
            resource="Magento_Catalog::catalog"/>

        <add id="Magento_Catalog::catalog_products"
            title="Catalog"
            module="Magento_Catalog"
            sortOrder="10"
            parent="Magento_Catalog::inventory"
            action="catalog/product/"
            resource="Magento_Catalog::products"/>

        ...
    </menu>
</config>
```

[HOME](#)**Magento U**

Notes:

Menu is configured by the file `menu.xml`.

The code example shows how to typically add a new menu item, using an “add” directive. You should specify its name (`id` attribute), the module it belongs to, a visible `title`, and an ACL (`resource` attribute).

6.18 Reinforcement Exercise (6.6.2)

Reinforcement Exercise (6.6.2): Menu

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

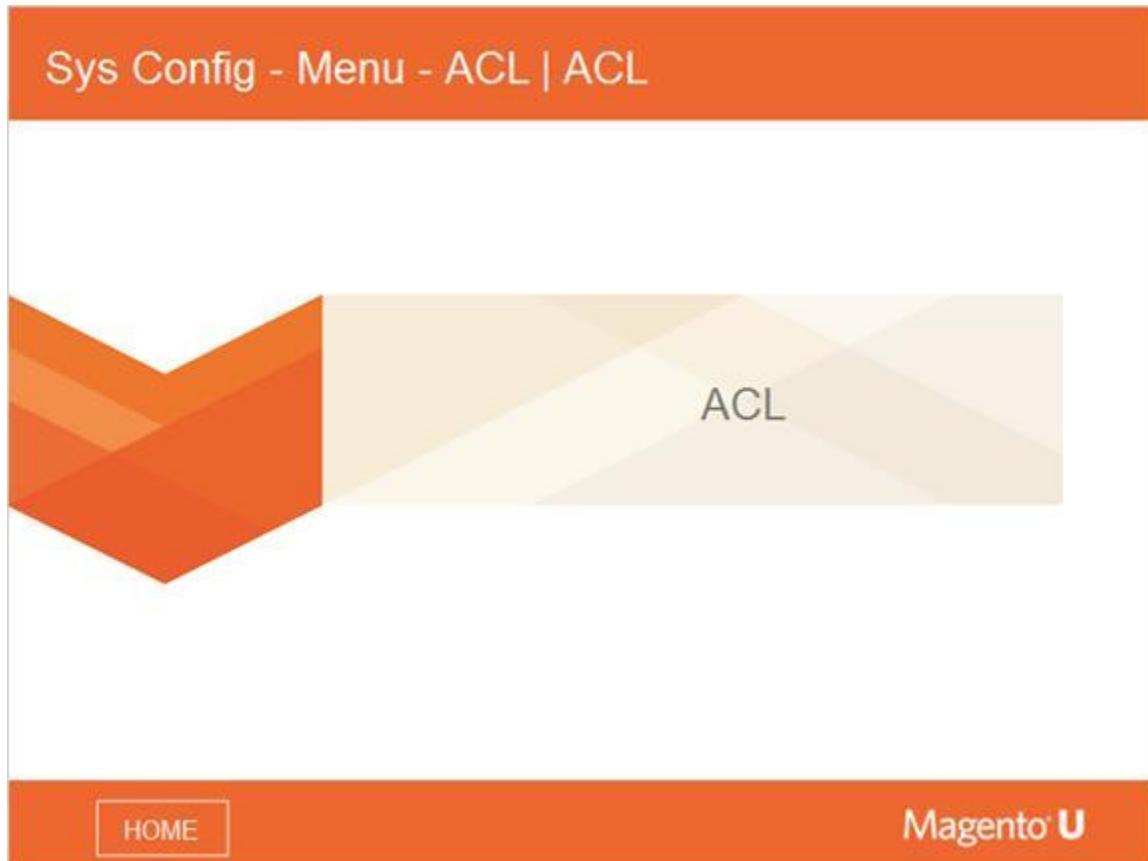
Click "Next" when done

[HOME](#) **Magento U**

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

6.19 Sys Config - Menu - ACL | ACL



Notes:

6.20 ACL | Definition

ACL | Definition

ACL in Magento 2 provides...

- An access restriction mechanism
- Access to each page defined as an ACL resource
- An ACL check, implemented in the `_isAllowed` method

[HOME](#) **Magento U**

Notes:

The ACL system in Magento 2 is very similar to Magento 1.

It has a following structure:

- Each admin user is assigned a certain ACL role.
- Each role is assigned access to a specific set of resources.
- As in Magento 1, ACL verification is implemented on the controller level, using the `_isAllowed` method.

6.21 ACL | _isAllowed Method

ACL | _isAllowed Method

Magento\Backend\App\AbstractAction

```
protected function _isAllowed()
{
    return $this->_authorization->isAllowed(self::ADMIN_RESOURCE);
}
```

[HOME](#)

Magento U

Notes:

The code presents an example of an `_isAllowed()` method.

You must implement this method for ACL verification to work.

6.22 ACL | _isAllowed Method Implementation Example

ACL | _isAllowed Method Implementation Example

Magento\Catalog\Controller\Adminhtml\Product

```
protected function _isAllowed()
{
    return $this->_authorization->isAllowed('Magento_Catalog::products');
}
```

[HOME](#)

Magento U

Notes:

This code example is taken from the Magento/Catalog module.

6.23 ACL | Users

The screenshot shows the 'ACL | Users' section of the Magento Admin interface. On the left, there's a sidebar with 'USER INFORMATION' and 'User Role' tabs. The main area has two tabs: 'User Info' and 'Account Information'. The 'User Info' tab is active, showing fields for User Name (admin), First Name (admin), Last Name (admin), Email (admin@test.magento.com), New Password (empty), Password Confirmation (empty), and Interface Locale (English (United States)). The 'Account Information' tab is shown in a modal window. It contains fields for User Name (admin), First Name (admin), Last Name (admin), Email (admin@test.magento.com), New Password (empty), Password Confirmation (empty), and Interface Locale (English (United States) / English (United States)). At the bottom of the main screen, there's a 'Current User Identity Verification' section with a 'Your Password' field. The footer features a 'HOME' button and the 'Magento U' logo.

Notes:

This image shows the admin user creation and management screen.

6.24 ACL | Roles

The screenshot shows the 'ACL | Roles' configuration page in the Magento admin. The main area displays a hierarchical tree of resources under 'Role Resources'. The tree includes categories like Dashboard, Sales, Operations, Orders, Actions, Invoices, Credit Memos, Billing Agreements, and Products, each with further sub-options such as Create, View, Send Order Email, Reorder, Edit, Cancel, Accept or Deny Payment, Capture, and Invoice.

On the left, there's a sidebar with tabs for 'Role Info', 'Resource Access' (set to 'Custom'), 'Role Resources', and 'Role Users'. A 'HOME' button is located at the bottom left of the page, and the 'Magento U' logo is at the bottom right.

Notes:

Here is an ACL roles tree. The list of available resources is defined in an `acl.xml` configuration file.

6.25 ACL | Configuration (Magento/Catalog/etc/acl.xml)

ACL | Configuration (Magento/Catalog/etc/acl.xml)

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance ...>
  <acl>
    <resources>
      <resource id="Magento_Backend::admin">
        <resource id="Magento_Catalog::catalog" title="Products" sortOrder="30">
          <resource id="Magento_Catalog::catalog_inventory" title="Inventory" sortOrder="10">
            <resource id="Magento_Catalog::products" title="Catalog" sortOrder="10" />
            <resource id="Magento_Catalog::categories" title="Categories" sortOrder="20" />
          </resource>
        </resource>
      <resource id="Magento_Backend::stores">
        <resource id="Magento_Backend::stores_settings">
          <resource id="Magento_Config::config">
            <resource id="Magento_Catalog::config_catalog" title="Catalog Section" />
          </resource>
        </resource>
        <resource id="Magento_Backend::stores_attributes">
          <resource id="Magento_Catalog::attributes_attributes" title="Product" sortOrder="30" />
          <resource id="Magento_Catalog::update_attributes" title="Update Attributes" sortOrder="35"/>
          <resource id="Magento_Catalog::sets" title="Attribute Set" sortOrder="40"/>
        </resource>
      </resource>
    </resources>
  </acl>
</config>
```

[HOME](#)**Magento U****Notes:**

This code provides an example of an acl.xml from the Magento/Catalog module.

6.26 Reinforcement Exercise (6.6.3)

Reinforcement Exercise (6.6.3): ACL

*See your course Exercises Guide for instructions
on how to complete this exercise, and its solution.*

Click "Next" when done

HOME

Magento U

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

6.27 End of Unit Six

End of Unit Six

Congratulations!
You have completed Unit Six
and now the entire
Fundamentals of Magento 2 Development course.

[HOME](#)

Magento U